A Moving-Horizon Approximate Branch-and-Reduce Method for Deep Classification Trees

Anonymous authors
Paper under double-blind review

Abstract

Despite the importance for interpretability, decision trees face severe scalability challenges. Existing global optimal methods are often limited by binary feature selection and shallow tree depths, whereas traditional heuristic approaches frequently sacrifice predictive accuracy. To overcome these limitations, this paper proposes a moving-horizon approximate branch-and-reduce method to train near-optimal deep classification trees on large-scale datasets with continuous features. Built on a bilevel optimization framework, the method solves the upper-level problem via branch-and-reduce while approximating the lower-level problem using greedy heuristics. Although the underlying framework is capable of guaranteeing global optimality, the approximation, which functions as a lookahead rollout in a reinforcement learning context, significantly boosts efficiency for deeper structures. A low-cost moving-horizon strategy is then employed to iteratively refine model accuracy. Extensive numerical results demonstrate that our method exceeds the testing accuracy of existing heuristic baselines while offering significantly greater scalability, in terms of both dataset size and tree depth, than global optimal solvers.

1 Introduction

The decision tree (DT) model is a cornerstone of machine learning, lauded for its interpretable, flowchart-like structure that makes it particularly suitable for classification and regression tasks requiring transparency and comprehensibility (Freitas, 2014; Krzywinski & Altman, 2017). Unlike many black-box models, DTs offer a higher degree of trustworthiness, which is critical in advancing AI for scientific research and high-stakes decision-making (Rudin, 2019; 2022). However, learning an optimal decision tree (ODT) has been classified to be $\mathcal{NP}$-hard (Laurent & Rivest, 1976). Traditional algorithms such as `CART` (Breiman et al., 1984), `ID3` (Quinlan, 1986), and `C4.5` (Quinlan, 1993) have been widely used due to their simplicity and efficiency, employing greedy heuristics to recursively partition data from the root to the leaves. Although these approaches generate trees efficiently, they fall significantly short of achieving optimality, especially for deeper trees.

Recent advancements in deterministic optimization techniques for learning ODTs have focused on approaches leveraging mixed-integer programming (MIP), satisfiability (SAT) solvers (Hu et al., 2020; Alòs et al., 2023), and dynamic programming (DP) approaches (Nijssen & Fromont, 2007; Van der Linden et al., 2022; van der Linden et al., 2023). Among these, MIP-based methods (Günlük et al., 2021; Zhu et al., 2020; Aghaei et al., 2024) formulate the training process as a mixed-integer optimization problem (Bertsimas & Dunn, 2017) and have achieved notable success by providing global optimization guarantees (optimality gap). However, efficiency remains a major limitation of these methods. For example, `Quant-BnB` and `RS-OCT` Mazumder et al. (2022); Hua et al. (2022) have demonstrated the ability to generate optimal trees up to depth 3, extending these approaches to deeper trees remains computationally hard. In contrast, some other scalable DP methods, such as `DL8.5` (Aglin et al., 2020) and `MurTree` (Demirović et al., 2022), leverage advanced caching strategies to enable the learning of relatively deeper trees. Despite these advances, most DP- and SAT-based approaches (Huisman et al., 2024; Lin et al., 2020; Avellaneda, 2020) face inherent drawbacks. They often require binarizing continuous features, which inflates the feature numbers and necessitates approximation techniques from `BinOCT` (Verwer & Zhang, 2019) (some methods directly consider encoding DTs (Shati et al., 2023)).

As a result, these methods solve a feature-binarized approximation of the original problem, which causes information loss, and they still struggle to build trees deeper than 5 on large datasets.

Beyond deterministic approaches, several heuristic methods warrant consideration. One notable example is `TAO` (Carreira-Perpinán & Tavallali, 2018; Carreira-Perpiñán & Zharmagambetov, 2020; Zharmagambetov et al., 2021), which has been reported to yield only modest improvements over `CART` (Zhu & Shoaran, 2021; Mazumder et al., 2022). More recently, `DPDT` (Kohler et al., 2025) has been proposed, which frames tree induction as a Markov Decision Process in conjunction with `CART`. Although `DPDT` has been shown to outperform `CART`, its further accuracy gains come at a substantial computational cost, which significantly limits its optimality. Within practical time budgets, the trees it produces remain far less accurate than globally optimal ones. This gap highlights the urgent need for an advanced method that can deliver both high accuracy and scalability for deep decision trees on large-scale datasets.

This paper introduces the Moving-Horizon Approximate Branch-and-Reduce (`MHABR`) algorithm for training deep classification trees on large-scale datasets with continuous features. This method is designed to achieve better optimization quality than heuristic methods (e.g., `CART`) while remaining substantially more scalable than global optimal solvers. Extensive numerical experiments show that `MHABR` can successfully train trees of depth 8 on datasets with up to 60 million samples, achieving stronger optimization quality than the heuristic baselines we evaluate, while scaling to deeper trees and larger datasets than the global methods considered in our experiments. Additionally, we provide an α-tuning option to mitigate overfitting, along with an extended variant that further improves optimality, closer to the global optimum.

Contributions: (1) We propose a bilevel optimization framework for learning DTs, where the root node parameters are optimized at the upper level, and the subtree parameters are optimized at the lower level. **(2)** Within this framework, we introduce a branch-and-reduce (`BR`) method for the upper-level problem. When the lower-level problem is solved recursively, the framework guarantees convergence to global optimality. **(3)** To improve efficiency for deep trees, we approximate the lower-level problem using `CART`. Under this approximate framework, exactness is retained for depth-2 trees because the depth-1 lower-level subproblems are solved exactly. **(4)** We introduce a moving-horizon (MH) technique to iteratively refine branch parameters at low cost, enabling the training of deep trees on large-scale datasets.

Performance: We evaluate the proposed method on 59 UCI datasets (Dua & Graff, 2017), with sample sizes ranging from 47 to 60,807,600 and tree depths from 2 to 8, and compare it against six baselines.

- **Testing accuracy:** On 51 small datasets (fewer than 10K samples), `MHABR` achieves the highest average testing accuracy among the compared methods at depths 2 and 3, excluding datasets for which `Quant-BnB` does not converge, and exceeds `DL8.5` by 0.99% on average across the reported depths. On the 8 medium- and large-scale datasets, `MHABR` improves average testing accuracy over `DPDT` by 3.01% at depth 8.
- **Scalability:** On 5 medium datasets (10K–1M samples) and 3 large datasets (1M–60M samples), `MHABR` remains computationally feasible at depth 8, whereas the compared global optimal methods do not complete within the specified time budget. Across these datasets, `MHABR` improves average test accuracy over `CART` by 3.66%.
- **Optimality:** `MHABR` guarantees global optimality for depth-2 trees. On the 42 small datasets completed by `Quant-BnB` at depth 3, its average training-accuracy gap to the global optimum is 0.58%.

2 Preliminary

This paper focuses on training a DT model for a classification task. The notation primarily follows Mazumder et al. (2022). We address a supervised learning problem involving a given dataset $\mathcal{X} = \{x_i \mid x_i \in \mathbb{R}^p, i \in [n]\}$ and corresponding label set $\mathcal{Y}$ that includes n_c classes, where $[n]$ denotes $\{1, \cdots, n\}$ and n is the number of samples, p is the number of features, x_i represents the feature vector that can admit numerical, categorical, or binary values. Initially, we scale each feature value of the dataset to the range $[0,\ 1]$.

2.1 Notations for Decision Trees

We develop a balanced binary tree formulation with axis-aligned splits and a fixed depth, denoted by d. Let $\mathcal{N}_B := \{1, \cdots, 2^d - 1\}$ and $\mathcal{N}_L := \{2^d, \cdots, 2^{d+1} - 1\}$ denote the sets of branch nodes and leaf nodes, respectively. In these trees, each branch node performs a logical test $x_i^a \leq b$, where x_i^a denotes the value of the feature a for the ith sample, thereby partitioning the samples into the left (if "yes") and right (if "no") subtrees. The splitting rule at each branch node is defined by two parameters: the feature indices and the corresponding splitting thresholds, collected in the vectors $\mathbf{a}, \mathbf{b}$, respectively. The partitioning operation is repeated at each layer until every sample is assigned to a leaf node. Each leaf node represents a specific region of the input space and holds a prediction, collected in the vector $\mathbf{c}$.

The induction of a decision tree $T : [0,1]^p \to \mathcal{Y}$ for a group of samples indexed by the set $\mathcal{I}$ can be formulated as an optimization problem expressed as

$$T^*(\mathcal{I}) = \arg\min_{T \in \mathcal{T}_d} L(\mathcal{I}, T), \quad \text{and} \quad V_d(\mathcal{I}) := \min_{T \in \mathcal{T}_d} L(\mathcal{I}, T), \tag{1}$$

where $\mathcal{T}_d$ represents the family of DT with depth d , $L(\mathcal{I}, T) := \sum_{i \in \mathcal{I}} \ell(y_i, T(x_i))$ denotes the number of misclassified instances associated with tree T over the samples $\mathcal{I}$, and $\ell := \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ denotes the loss function. For classification tasks in this paper, the loss function is defined as $\ell(y_i, \hat{y}_i) = \mathbb{1}\{y_i \neq \hat{y}_i\}$. In this formulation, a penalty term $\alpha \cdot \mathcal{C}$ can be introduced to balance the trade-off between training accuracy and model generalization, where $\mathcal{C}$ denotes the complexity of the decision tree model and α serves as the regularization coefficient. However, this paper focuses on optimizing the training process; thus, the penalty term is omitted in this formulation. The term will later be incorporated into numerical experiments to assess its impact on model performance.

For a selected feature a, let n_a denote the number of unique values in the dataset $x_i^a,\ i \in \mathcal{I}^a$, where $\mathcal{I}^a := [n]$ represents the index set of the data sorted by the values of feature a. Define $\mathcal{K}^a := [n_a + 1]$ as the index set of distinct values, denoted as $u_1^a < u_2^a < \cdots < u_{n_a}^a$, with boundary points $u_0^a = u_1^a - 1$ and $u_{n_a+1}^a = u_{n_a}^a + 1$. Since all points within the interval (u_{k-1}^a, u_k^a) are equivalent for splitting, the splitting threshold is defined as $\beta_k^a := \frac{u_{k-1}^a + u_k^a}{2}$ for each $k \in \mathcal{K}^a$. We define $\zeta : \mathcal{K}^a \to \mathcal{I}^a$ such that $\zeta(k)$ is the index of the first occurrence of the unique element u_k^a in the sequence $(x_i^a)_{i \in \mathcal{I}^a}$. For the root node, it suffices to consider candidate trees with all splits located in the set $\mathcal{B} := \{\beta_k^a | a \in [p], k \in \mathcal{K}^a\}$. Given a set $\mathcal{I}^a$ and two integers k_1 and k_2 such that $0 \leq k_1 \leq k_2 \leq n_a$, we define

$$\mathcal{I}_{[k_1,k_2]}^a := \{i \in \mathcal{I}^a \mid \beta_{k_1}^a \leq x_i^a \leq \beta_{k_2}^a\}, \tag{2}$$

as the set of sample indices i for which the corresponding feature values x_i^a fall within $[\beta_{k_1}^a,\ \beta_{k_2}^a]$.

2.2 A Bilevel Framework for Decision Tree Training

We propose a bilevel framework for training decision trees based on the above notations. We denote $T := [a, k, T_L, T_R]$, where a denotes the feature selected at the root node, k denotes the index of the root split among the candidate thresholds for feature a (that is, the threshold is β_k^a), and the parameters of left and right subtrees T_L and T_R. The root node typically exerts the most significant influence on the overall loss in a decision tree (Dwyer & Holte, 2007; Shannon & Banks, 1999), as it processes the entire dataset due to its position at the top of the data flow. Consequently, we isolate the parameters of the root node from the remainder of the tree for optimization through an upper-level problem (ULP). The optimal loss is subsequently defined as the sum of the losses from the two subtrees: T_L and T_R, optimized in the lower-level problem (LLP). Then, the optimization problem for training a depth-d tree T can be denoted as:

$$V_d(\mathcal{I}) := \min_{\substack{a \in [p],\ k \in \mathcal{K}^a \\ T_L, T_R \in \mathcal{T}_{d-1}}} L(\mathcal{I}, [a, k, T_L, T_R]) = \min_{a \in [p],\ k \in \mathcal{K}^a} \left\{ V_{d-1}(\mathcal{I}_{[0,k]}^a) + V_{d-1}(\mathcal{I}_{[k,n_a]}^a) \right\}, \tag{3}$$

as well as an explicit bilevel programming problem:

$$\min_{\substack{a \in [p] \\ k \in \mathcal{K}^a}} \left\{ \min_{T_L \in \mathcal{T}_{d-1}} L(\mathcal{I}_{[0,k]}^a, T_L) + \min_{T_R \in \mathcal{T}_{d-1}} L(\mathcal{I}_{[k,n_a]}^a, T_R) \right\}, \tag{4}$$

where the samples split to the left and right subtrees are denoted by $\mathcal{I}^a_{[0,k]}$ and $\mathcal{I}^a_{[k,n_a]}$. In addition, we use a similar notation to define the optimal value when a is fixed, or when a and k are both fixed.

$$V_d(\mathcal{I}, a) = \min_{k \in \mathcal{K}^a} V_d(\mathcal{I}, a, k) = \min_{k \in \mathcal{K}^a} \left\{ V_{d-1}(\mathcal{I}^a_{[0,k]}) + V_{d-1}(\mathcal{I}^a_{[k,n_a]}) \right\}. \tag{5}$$

3 A Branch-and-Reduce Method for Optimal Decision Trees

The bilevel framework, which is explored in BR method (detailed in Section 3.2), forms the basis of the approach discussed in this section. Here, we present this approach for solving *ULP*, under the key assumption that *LLP* can be solved exactly and efficiently.

Within the *ULP*, the decision variables are a and k. The variable $a \in [p]$ can be readily enumerated given the typically limited number of features. However, the selection of $k \in \mathcal{K}^a$ is more complex, contingent upon the number of unique feature values n_a. For continuous features, n_a can be as large as n, rendering the search space prohibitively extensive for exhaustive enumeration. To mitigate this challenge, we propose the BR method to determine the optimal k in this section.

3.1 Upper Bound, Lower Bound, Reduction, and Branching Strategy

The BR method involves four primary steps: Upper Bound, Lower Bound, Reduction Strategy, and Branching Strategy. We now present their details in a general form. Consider a node within the BR process, represented by the set of potential split indices $\mathcal{K}_i = \{k \mid l \le k \le m\}$ $(l, m \in [n_a])$. In the following description, we specifically select the midpoint $\bar{k} = \lceil \frac{l+m}{2} \rceil$ (the midpoint of $\mathcal{K}_i$) for branching.

Upper Bound The upper bound U denotes the best loss among previously evaluated splits, serving as a historical record throughout the iterative search within the feasible set. It is updated iteratively after evaluating a selected index $\bar{k}$ by:

$$U \leftarrow \min\{U,\ V_d(\mathcal{I}, a, \bar{k})\}, \quad \text{where } V_d(\mathcal{I}, a, \bar{k}) = V_{d-1}(\mathcal{I}^a_{[0,\bar{k}]}) + V_{d-1}(\mathcal{I}^a_{[\bar{k},n_a]}). \tag{6}$$

Boundary Analysis Define $L(\mathcal{I})$ as the optimal loss of the dataset indexed by $\mathcal{I}$ when training a DT of a fixed depth:

$$L(\mathcal{I}) = \min_{T \in \mathcal{T}} L(\mathcal{I}, T). \tag{7}$$

The following lemma bounds the loss function L in Equation (1).

Lemma 3.1. *For two sample index set $\mathcal{I}_1$ and $\mathcal{I}_2$ with $\mathcal{I}_1 \subseteq \mathcal{I}_2$, let n_1 and n_2 be the element numbers of $\mathcal{I}_1$ and $\mathcal{I}_2$, where $n_2 \ge n_1$, we have*

$$0 \le L(\mathcal{I}_2) - L(\mathcal{I}_1) \le n_2 - n_1.$$

Proof. From the formulation of Equation (1), we have:

$$\begin{aligned} L(\mathcal{I}_1) = \min_{T \in \mathcal{T}} \sum_{i \in \mathcal{I}_1} \ell(y_i, T(x_i)) &\le \min_{T \in \mathcal{T}} \sum_{i \in \mathcal{I}_1 \cup \{\mathcal{I}_2 \setminus \mathcal{I}_1\}} \ell(y_i, T(x_i)) \\ &= \min_{T \in \mathcal{T}} \sum_{i \in \mathcal{I}_2} \ell(y_i, T(x_i)) = L(\mathcal{I}_2). \end{aligned} \tag{8}$$

So, we have $L(\mathcal{I}_1) \le L(\mathcal{I}_2)$, which implies that prediction errors monotonically increase with sample numbers. Suppose the optimal tree for $\mathcal{I}_1$ is T^*_1, and for $\mathcal{I}_2$ we have

$$\begin{aligned} L(\mathcal{I}_2) \le L(\mathcal{I}_2, T^*_1) &= \sum_{i \in \mathcal{I}_2} \ell(y_i, T^*_1(x_i)) \\ &= \sum_{i \in \mathcal{I}_1} \ell(y_i, T^*_1(x_i)) + \sum_{i \in \mathcal{I}_2 \setminus \mathcal{I}_1} \ell(y_i, T^*_1(x_i)) \\ &= L(\mathcal{I}_1) + \sum_{i \in \mathcal{I}_2 \setminus \mathcal{I}_1} \ell(y_i, T^*_1(x_i)), \end{aligned} \tag{9}$$

where the first inequality is because $L(\mathcal{I}_2)$ is the optimal value. Then we get:

$$L(\mathcal{I}_2) - L(\mathcal{I}_1) \leq \sum_{i \in \mathcal{I}_2 \setminus \mathcal{I}_1} \ell(y_i, T_1^*(x_i)) \leq \sum_{i \in \mathcal{I}_2 \setminus \mathcal{I}_1} \mathbf{1} = n_2 - n_1. \tag{10}$$

Now we have $0 \leq L(\mathcal{I}_2) - L(\mathcal{I}_1) \leq n_2 - n_1$, which bounds the loss function L. □

Lower Bound The lower bound is based on a Lipschitz-type condition, which is illustrated by:

Lemma 3.2. *For any $\mathcal{I} \subseteq [n]$ and $d \geq 1$, let k_1 and k_2 be two split indices corresponding to feature a. Then we have $|V_d(\mathcal{I}, a, k_1) - V_d(\mathcal{I}, a, k_2)| \leq |\zeta(k_1) - \zeta(k_2)|$.*

Proof. Suppose $0 \leq k_1 \leq k_2 \leq n_a$, and we define $L(\mathcal{I})$ as Equation (7). From the Equation (3), we have

$$V_d(\mathcal{I}, a, k_1) = L(\mathcal{I}^a_{[0,k_1]}) + L(\mathcal{I}^a_{[k_1,n_a]}), \tag{11}$$

$$V_d(\mathcal{I}, a, k_2) = L(\mathcal{I}^a_{[0,k_2]}) + L(\mathcal{I}^a_{[k_2,n_a]}). \tag{12}$$

According to Lemma 3.1, we have

$$\begin{aligned} V_d(\mathcal{I}, a, k_2) - V_d(\mathcal{I}, a, k_1) &= L(\mathcal{I}^a_{[0,k_2]}) + L(\mathcal{I}^a_{[k_2,n_a]}) - L(\mathcal{I}^a_{[0,k_1]}) - L(\mathcal{I}^a_{[k_1,n_a]}) \\ &= \{L(\mathcal{I}^a_{[0,k_2]}) - L(\mathcal{I}^a_{[0,k_1]})\} - \{L(\mathcal{I}^a_{[k_1,n_a]}) - L(\mathcal{I}^a_{[k_2,n_a]})\} \\ &\leq \zeta(k_2) - \zeta(k_1). \end{aligned} \tag{13}$$

We take the case where $0 \leq k_2 \leq k_1 \leq n_a$ into consideration, which can be easily proven by reversal. Consequently, we get $|V_d(\mathcal{I}, a, k_1) - V_d(\mathcal{I}, a, k_2)| \leq |\zeta(k_1) - \zeta(k_2)|$. □

From Lemma 3.2, for a given loss of $\bar{k}$, the lower bound of $\mathcal{K}_i$ is

$$LB := \min\{V_d(\mathcal{I}, a, \bar{k}) - |\zeta(\bar{k}) - \zeta(l)|, V_d(\mathcal{I}, a, \bar{k}) - |\zeta(\bar{k}) - \zeta(m)|\} \tag{14}$$

Reduction Strategy Given $V_d(\mathcal{I}, a, \bar{k})$, the reduction strategy aims to reduce the search space by removing $\Delta = \{k \in \mathcal{K}_i \mid V_d(\mathcal{I}, a, k) > U\}$. The following lemma establishes a reduction strategy in which the candidate set is updated as $\mathcal{K}_i \leftarrow \mathcal{K}_i \setminus \Delta$.

Lemma 3.3. *For any $\mathcal{I} \subseteq [n]$ and $d \geq 1$, let $V_d(\mathcal{I}, a, \bar{k})$ be a loss value of Problem (3) corresponding to the split rule $\beta^a_{\bar{k}}$, and an upper bound U (assume $U \leq V_d(\mathcal{I}, a, \bar{k})$) of this problem. Define $\delta(\bar{k}) := V_d(\mathcal{I}, a, \bar{k}) - U$. Then, for any $|\zeta(\bar{k}) - \zeta(k)| \leq \delta(\bar{k})$, we have $V_d(\mathcal{I}, a, k) \geq U$; so, the optimal split index is not in $\Delta = \{k \mid |\zeta(\bar{k}) - \zeta(k)| \leq \delta(\bar{k}),\ k \in \mathcal{K}_i\}$.*

Proof. Suppose there is a split $k' \in \Delta$ such that $V_d(\mathcal{I}, a, k') < U$. According to Lemma 3.2, we have

$$|V_d(\mathcal{I}, a, \bar{k}) - V_d(\mathcal{I}, a, k')| \leq |\zeta(\bar{k}) - \zeta(k')| \leq \delta = V_d(\mathcal{I}, a, \bar{k}) - U. \tag{15}$$

Since $V_d(\mathcal{I}, a, k') < U \leq V_d(\mathcal{I}, a, \bar{k})$, we have

$$V_d(\mathcal{I}, a, \bar{k}) - V_d(\mathcal{I}, a, k') > V_d(\mathcal{I}, a, \bar{k}) - U > 0, \tag{16}$$

which is a contradiction of Equation (15). So there does not exist a split in Δ with a loss value less than U, i.e., the optimal solution is not in Δ. □

Branching Strategy To better expand Δ, the branching strategy bisects the candidate split set $\bar{k}$ at its **midpoint**, ensuring a balanced and effective reduction (Ryoo & Sahinidis, 1996; Tawarmalani & Sahinidis, 2004). We can divide $\mathcal{K}_i$ by the midpoint $\bar{k}$ into two subsets: $\mathcal{K}_L$ and $\mathcal{K}_R$ after reduction.

3.2 The Procedure of Branch-and-Reduce Method

Now, we are ready to introduce the procedure of our BR method. This method is designed to learn globally optimal depth-d decision trees. The process begins with an initial estimate (e.g., using CART) of the upper bound U and the dataset $\mathcal{I}$. For $a \in [p]$, the algorithm sorts the samples by their feature values to obtain $\mathcal{I}^a$ and constructs the candidate split set $\mathcal{B}^a$. The corresponding index set $\mathcal{K}^a$ contains all split candidates β_k^a for $k \in \mathcal{K}^a$, which is then passed to the main loop to obtain the optimal k.

Algorithm 1 Branch-and-reduce method (BR)

function BR($\mathcal{I}, d$):
  Initialize U;
  for $a \in [p]$ **do**
    Sort $\mathcal{I}$ to obtain $\mathcal{I}^a$, $\mathcal{K}^a$, and set $\mathbb{K} \leftarrow \{\mathcal{K}^a\}$;
    $U, T \leftarrow$ BnR($U, T, \mathcal{I}, d, \mathbb{K}$); *# solve for optimal trees*
return: Optimal loss U and decision tree T.

function BnR($U, T, \mathcal{I}, d, \mathbb{K}$):
  Initialize the iteration index $i = 0$;
  while $\mathbb{K} \neq \emptyset$ **do**
    node selection
      Select a set $\mathcal{K}_i$ from $\mathbb{K}$, set $\mathbb{K} \leftarrow \mathbb{K} \backslash \mathcal{K}_i$, and update $i \leftarrow i + 1$;
    upper bound
      Select a point $\bar{k} \in \mathcal{K}_i$, obtain $\mathcal{I}^a_{[0,\bar{k}]}$, $\mathcal{I}^a_{[\bar{k},n_a]}$;
      $(V_{d-1}(\mathcal{I}^a_{[0,\bar{k}]}),\ T_L) \leftarrow$ BR($\mathcal{I}^a_{[0,\bar{k}]},\ d-1$), $(V_{d-1}(\mathcal{I}^a_{[\bar{k},n_a]}), T_R) \leftarrow$ BR($\mathcal{I}^a_{[\bar{k},n_a]}, d-1$);
      Evaluate loss $V_d(\mathcal{I}, a, \bar{k}) = V_{d-1}(\mathcal{I}^a_{[0,\bar{k}]}) + V_{d-1}(\mathcal{I}^a_{[\bar{k},n_a]})$;
      if $V_d(\mathcal{I}, a, \bar{k}) < U$ **then**
        $U \leftarrow V_d(\mathcal{I}, a, \bar{k})$, update T;
    lower bound
      Calculate LB by Equation (14), and obtain the optimality gap $U - LB$;
    if $U - LB \leq$ tol **then**
      break
    reduction
      $\delta(\bar{k}) \leftarrow V_d(\mathcal{I}, a, \bar{k}) - U$;
    branching
      Obtain $\mathcal{K}_L$, $\mathcal{K}_R$ by $\bar{k}$. Update $\mathbb{K} \leftarrow \mathbb{K} \cup \{\mathcal{K}_L\}$, if $\mathcal{K}_L \neq \emptyset$, $\mathbb{K} \leftarrow \mathbb{K} \cup \{\mathcal{K}_R\}$, if $\mathcal{K}_R \neq \emptyset$;
return: Optimal loss U and decision tree T.

The main loop of the BR method solves for $V_d(\mathcal{I}, a)$ with fixed a. At each iteration, the steps of this loop are as follows: **(i)** A BR node containing the set $\mathcal{K}_i$ is selected from the entire feasible region of k, a step referred to as BR node selection. **(ii)** The midpoint candidate split $\bar{k}$ is then chosen from $\mathcal{K}_i$. The algorithm evaluates $V_d(\mathcal{I}, a, \bar{k})$, and updates the upper bound if the new solution improves upon the current best result. **(iii)** Computing $\delta(\bar{k})$ by the updated upper bound U, the reduction strategy narrows down the set $\mathcal{K}_i$. **(iv)** Once the reduced set is determined, branching occurs on the remaining region. While the lower bound from Equation (14) enables the calculation of the optimality gap and termination based on a specified tolerance, to ensure a fair comparison with state-of-the-art methods, the subsequent method employs a complete search strategy with a tolerance of 0. Therefore, we consider terminating the main loop when $\mathbb{K} = \emptyset$. We can easily prove that Algorithm 1 converges to the **global optimum** of Equation (3) based on Lemmas 3.2 and 3.3.

Theorem 3.4. *Algorithm 1 converges to the global optimum of Equation* (3).

Proof. Since Equation (3) is a bilevel optimization problem, we analyze both *ULP* and *LLP*. We begin with the case of a depth-1 tree (including R-CART), where the *LLP* corresponds to optimizing a constant fit. To establish the optimality of Algorithm 1 for a depth-1 tree, it suffices to show that the reduction strategy does not exclude any optimal solution, as the algorithm without reduction effectively performs an exhaustive

enumeration. Lemma 3.3 demonstrates that the optimal split is never removed from the reduced set Δ, thereby ensuring that Algorithm 1 converges to the optimal solution of the *LLP*.

Next, we assume that Algorithm 1 converges to the global optimal solution for a depth-$(d-1)$ tree. For a depth-d tree, each *LLP* subproblem corresponds to optimizing a depth-$(d-1)$ tree, which, by assumption, is solved to global optimality. The *ULP* follows the same proof structure as the depth-1 case. Consequently, by induction, Algorithm 1 converges to the global optimum of Equation (3). □

Remark 3.5. The reduction strategy is the main source of efficiency gains in this algorithm, with a trivial subtraction operation. It is similar to the first lower bounding problem proposed by Mazumder et al. (2022) but is generalized to any depth. The other lower bounding problems proposed by Hua et al. (2022); Mazumder et al. (2022) involve solving complicated lower bounding problems, leading to additional computational overhead. In contrast, BR requires only the computation of $\delta(\bar{k})$.

4 A Moving-Horizon Approximate Method for Deep Trees

In the above bilevel optimization framework, we assume that the calculation of V_{d-1} in *LLP* is solved exactly. Under this assumption, global convergence can be guaranteed for trees of any depth. This condition can be satisfied by recursively calling `BR` for *LLP*, but it results in an exponential increase in computational time. To address this limitation, this section introduces an approximate method that significantly improves computational efficiency while maintaining high accuracy. Despite the approximation, we show that the global optimality for depth-2 trees is still guaranteed.

4.1 An Approximate Problem with *LLP*

In tree structures, the influence of individual nodes on classification loss generally decreases with depth, as deeper nodes tend to affect fewer data points. This implies that *LLP* has a lesser impact on overall accuracy compared to *ULP*. Consequently, we employ the `CART` algorithm to obtain an approximate solution for *LLP*. This method offers computational efficiency in addressing the *LLP*, with limited impact on the optimality, but substantially reducing the overall computational demand. Let $W_d(\mathcal{I})$ denote the loss of a depth-d `CART` tree $\hat{T}$ on dataset $\mathcal{I}$. The `CART` splitting rule for each branch node typically involves minimizing the combined loss of the potential children (detailed in Section 8.1). In our implementation, we use the result of this approximate method to update the upper bound, where the approximate problem is denoted as

$$\hat{V}_d(\mathcal{I}) := \min_{a\in[p],k\in\mathcal{K}^a} \hat{V}_d(\mathcal{I},a,k) = \min_{a\in[p],k\in\mathcal{K}^a} \{W_{d-1}(\mathcal{I}^a_{[0,k]}) + W_{d-1}(\mathcal{I}^a_{[k,n_a]})\}. \tag{17}$$

Compared to `CART`, `ABR` performs a broader root-level search under approximate subtree evaluation, leading to improved accuracy, which is essential for the effectiveness of the following MH procedure. Drawing upon the procedural steps of `BR`, we can obtain an *approximate branch-and-reduce (`ABR`)* method to solve $\hat{V}_d(\mathcal{I})$, detailed in Algorithm 3. Then, we have the following lemma (detailed in Section 4.4), stating that **`ABR` is no worse than `CART`** in this case.

Lemma 4.1. *Suppose there is no reduction strategy in Algorithm 3, let $W_d(\mathcal{I})$ and $\hat{V}_d(\mathcal{I})$ be as Equation* (17)*, then $\hat{V}_d(\mathcal{I}) \le W_d(\mathcal{I})$ for all $d > 0$.*

Proof. Since there is no reduction, we consider a complete search of the range. From Equation (17), $\hat{V}_d(\mathcal{I})$ is calculated by solving

$$\begin{aligned}\hat{V}_d(\mathcal{I}) &= \min_{a\in[p],k\in\mathcal{K}^a} \hat{V}_d(\mathcal{I},a,k)\\ &= \min_{a\in[p],k\in\mathcal{K}^a} \{L(\mathcal{I}^a_{[0,k]},\hat{T}_L) + L(\mathcal{I}^a_{[k,n_a]},\hat{T}_R)\}\\ &= \min_{a\in[p],k\in\mathcal{K}^a} \{W_{d-1}(\mathcal{I}^a_{[0,k]}) + W_{d-1}(\mathcal{I}^a_{[k,n_a]})\}\end{aligned} \tag{18}$$

and $W_d(\mathcal{I})$ can be calculated as

$$W_d(\mathcal{I}) = \hat{V}_d(\mathcal{I},a',k') = W_{d-1}(\mathcal{I}^{a'}_{[0,k']}) + W_{d-1}(\mathcal{I}^{a'}_{[k',n_{a'}]}), \tag{19}$$

where (a', k') denotes the root node parameters obtained by `CART` (as well as an optimal depth-1 tree), and is calculated by:

$$\begin{aligned} a', k' &= \arg \min_{a\in[p], k\in\mathcal{K}^a} \left\{ V_0(\mathcal{I}^a_{[0,k]}) + V_0(\mathcal{I}^a_{[k,n_a]}) - V_0(\mathcal{I}) \right\} \\ &= \arg \min_{a\in[p], k\in\mathcal{K}^a} \left\{ V_0(\mathcal{I}^a_{[0,k]}) + V_0(\mathcal{I}^a_{[k,n_a]}) \right\} \\ &= \arg \min_{a\in[p], k\in\mathcal{K}^a} \left\{ \min_{c_1\in\mathcal{N}_c} L(\mathcal{I}^a_{[0,k]}, c_1) + \min_{c_2\in\mathcal{N}_c} L(\mathcal{I}^a_{[k,n_a]}, c_2) \right\} \end{aligned}$$

where $V_0(\mathcal{I}) = \min_{c\in\mathcal{N}_c} L(\mathcal{I}, c)$ can be easily calculated as $\mathcal{I}$ is given, and it represents the rate of the most frequent class. So $L(\mathcal{I}, (a', k')) = W_1(\mathcal{I})$ holds. It is obvious that (a', k') is a feasible solution in the range of Equation (18), while $\hat{V}_d(\mathcal{I})$ is the optimal value of Equation (18). Then we have $\hat{V}_d(\mathcal{I}) \leq W_d(\mathcal{I})$ for all $a \in [p], k \in \mathcal{K}^a$. So $\hat{V}_d(\mathcal{I}) \leq W_d(\mathcal{I})$, this lemma is proved. □

Global optimal case For depth-1 trees, the greedy method can identify global optimal splits, providing exact solutions for the lower-level subproblems in `ABR` while solving a depth-2 tree. Consequently, `ABR` guarantees **global optimality** for depth-2 trees, matching the performance of global optimal methods, as verified by numerical experiments.

4.2 A Reinforcement Learning Perspective

`ABR` can be regarded as an approximate DP method (Bertsekas, 2024), and can be explained by Reinforcement Learning (RL) theories. Following `DPDT` (Kohler et al., 2025), we model the DT problem as a Markov Decision Process. At each layer, the state is defined by the *set of samples* present at the nodes of that layer. The action at a given layer corresponds to the *set of splitting rules* applied to all nodes in that layer. The reward measures the *loss descent* achieved by the splits, calculated as the difference between the total loss at the previous layer and that at the current layer.

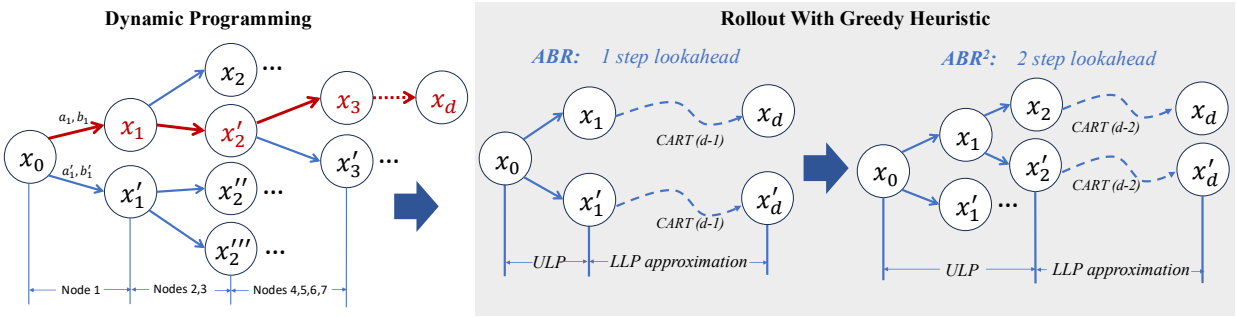


Figure 1: Illustration of the *LLP* Approximation via RL (x denotes the state).

For a depth-d tree, the overall objective is to select splitting rules at all layers to minimize the cumulative loss from the root to the leaves. This can be interpreted as a "cost-to-go" problem:

1st stage cost + optimal tail problem cost.

Figure 1 is an illustration of *LLP* approximation. Computing the global optimum requires DP that explores all feasible splits, whose number grows exponentially with tree depth. To make this tractable, *LLP* is approximated in the *value space* (see Section 1.2.3 of Bertsekas (2024)) by constructing a suboptimal policy from two *LLP* subtrees using heuristics such as `CART`, replacing the *optimal tail problem cost*. This approximation can be further enhanced by starting the heuristic from stage 2, a 2-step lookahead method, which improves performance at the cost of additional computation.

4.3 A Moving-Horizon Approach

The proposed approximation method uses `CART` to generate suboptimal solutions for the left and right subtrees when $d > 2$. To improve solution quality, we introduce the MH approach that iteratively refines branch parameters. At each node, the subsequent nodes form a subtree that is re-optimized with its corresponding samples $\mathcal{I}_{sub}$, creating a new optimization subproblem. As tree depth increases, this iterative refinement helps close the gap between `ABR` and the global optimum. As illustrated in Figure 2, the MH process begins at node 1, where `ABR` updates all branch nodes in the depth-4 tree ($T_{d=4}$, nodes $1, 2, \ldots, 15$). Next, with node 2 as the root, `ABR` refines the branch nodes of the depth-3 subtree ($T_{d=3}$, nodes $2, 4, 5, 8, 9, 10, 11$). This iterative procedure continues in each step, with the subtree depth decreasing until the subtree depth $d_{sub} = 2$. Here, nodes such as $4, 8, 9$ in $T_{d=2}$ are updated three times throughout this process.

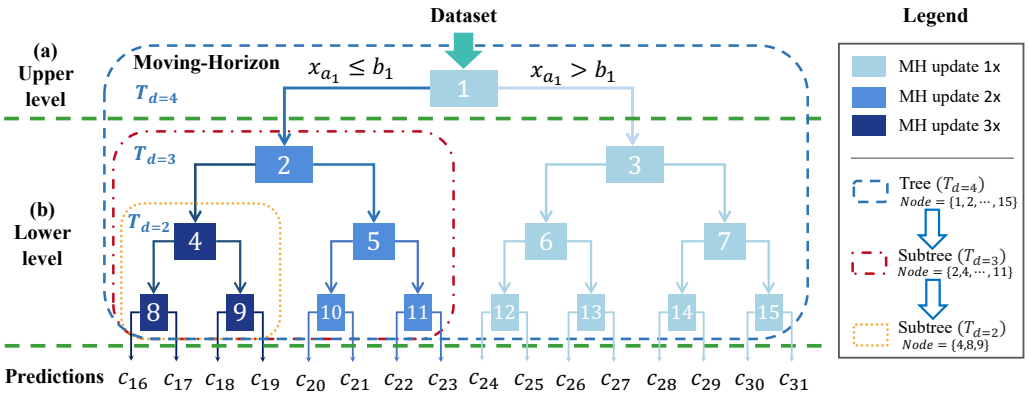


Figure 2: An example of MH when $d = 4$.

After applying the MH refinement at node $t \in \mathcal{N}_B$, we update the corresponding subtree parameters of T using T_{sub} as follows

$$T[t \cdot 2^j : (t+1) \cdot 2^j - 1] \Leftarrow T_{sub}[2^j : 2^{j+1} - 1], \quad j \in 0, \ldots, d - d_{sub}. \tag{20}$$

This iterative refinement can substantially improve the accuracy of deep decision trees, requiring at most $2^{d-1} - 1$ iterations at relatively low cost, and terminates when a subtree reaches an optimal loss, i.e., $d_{sub} = 1$ or $L(\mathcal{I}_{sub}, T_{sub}) = 0$.

4.4 Why `ABR` and MH Can Improve Over `CART`

We now give intuition for why the proposed approximation can outperform greedy `CART`. The key difference is that `CART` selects each split using a purely local criterion, whereas `ABR` evaluates a candidate root split by combining it with approximate subtree costs. As a result, `ABR` can prefer a split that appears suboptimal under a one-step greedy criterion but yields a better tree once the downstream subtree structure is considered. A simple illustrative example (detailed in Section 8.3) is the XOR problem (Guyon & Elisseeff, 2003), where a single greedy split may fail to reveal the informative feature interaction, while a depth-2 lookahead can recover the correct structure.

Although `ABR` uses a stronger root-level objective than `CART`, it still relies on approximate lower-level subtree values. The purpose of the MH refinement is to reduce the resulting approximation error by re-optimizing selected subtrees after the higher-level structure has been fixed. In this sense, MH can be viewed as a local repair mechanism: it revisits subproblems that were previously solved only approximately and may further reduce the training loss. For `CART`, the depth-d objective is

$$W_d(\mathcal{I}) = W_{d-1}(\mathcal{I}^{a'}_{[0,k']}) + W_{d-1}(\mathcal{I}^{a'}_{[k',n_{a'}]}), \tag{21}$$

where (a', k') is the greedy root split. In contrast, ABR selects the root split according to

$$\hat{V}_d(\mathcal{I}) = W_{d-1}(\mathcal{I}^{a^*}_{[0,k^*]}) + W_{d-1}(\mathcal{I}^{a^*}_{[k^*,n_{a^*}]}), \tag{22}$$

where (a^*, k^*) minimizes the approximate lookahead objective. Thus, ABR may select a different root split from CART, and the subsequent MH refinements can further improve the resulting tree by re-optimizing selected subtrees. We evaluate this effect empirically in the ablation study of Section 5.6.

4.5 A Moving-Horizon Approximate Branch-and-Reduce Method

Algorithm 2 Main Loop of ABR

function branch$(U, T, \mathcal{I}, d, \mathbb{K})$:
  Initialize the iteration index $i = 0$;
  while $\mathbb{K} \neq \emptyset$ **do**
    Node Selection:
    $\mathcal{K}_i$ from $\mathbb{K}$, set $\mathbb{K} \leftarrow \mathbb{K} \backslash \mathcal{K}_i$, update $i \leftarrow i + 1$;
    Select $\bar{k} \in \mathcal{K}_i$, obtain $\mathcal{I}^a_{[0,\bar{k}]}$, $\mathcal{I}^a_{[\bar{k},n_a]}$;
    Calculate $\hat{V}_d(\mathcal{I}, a, \bar{k})$ by Equation (17)
    Update Upper Bound:
    if $\hat{V}_d(\mathcal{I}, a, \bar{k}) < U$ **then**
      $U \leftarrow \hat{V}_d(\mathcal{I}, a, \bar{k})$, update T;
    Reduce: $\delta(\bar{k}) \leftarrow \max\{\hat{V}_d(\mathcal{I}, a, \bar{k}) - U, 1\}$;
    Branch: obtain $\mathcal{K}_L$, $\mathcal{K}_R$ by midpoint;
    Update $\mathbb{K} \leftarrow \mathbb{K} \cup \{\mathcal{K}_L\}$, if $\mathcal{K}_L \neq \emptyset$ and update $\mathbb{K} \leftarrow \mathbb{K} \cup \{\mathcal{K}_R\}$, if $\mathcal{K}_R \neq \emptyset$;
return: Loss U and decision tree T.

Algorithm 3 ABR (for subtrees)

function ABR$(\mathcal{I}, d)$:
  Initialize U and T with CART;
  for $a \in [p]$ **do**
    Sort $\mathcal{I}$ to obtain $\mathcal{I}^a$, $\mathcal{K}^a$, and set $\mathbb{K} \leftarrow \{\mathcal{K}^a\}$;
    Calculate $U, T \leftarrow$ branch$(U, T, \mathcal{I}, d, \mathbb{K})$;
return: Loss U and decision tree T.

Algorithm 4 MHABR

input Dataset $\mathcal{I}$, depth d, and initial upper bound U;
for $t \in [2^{d-1} - 1]$ **do**
  Get data $\mathcal{I}_t$ at tree node t;
  if $\mathcal{I}_t \neq \emptyset$ **then**
    Calculate $U_{sub}, T_{sub} \leftarrow$ ABR$(\mathcal{I}_t, d_{sub})$;
    Update U and T through Equation (20);
output Loss U and decision tree T.

By leveraging the *LLP* approximation and the MH technique, we present the MHABR algorithm for a depth-d tree, as described in Algorithm 4. Designed specifically for trees with $d \geq 2$, the algorithm applies the MH refinement for at most $2^{d-1} - 1$ iterations. At each iteration, it refines the current model by optimizing a selected subtree, using ABR (Algorithm 3) to yield a subtree solution T_{sub} with depth d_{sub} and objective value U_{sub}. Because ABR maintains and updates an incumbent solution over candidate thresholds within the branch subroutine, it naturally admits a warm-start strategy. In practice, warm-starting improves efficiency and ensures that the returned subtree solution is no worse than the CART initialization. We now analyze the computational complexity of the algorithm.

Theorem 4.2. *Suppose that each upper-level search considers at most $\tilde{n}$ split candidates and ignore implementation-dependent speedups such as warm-starting, reduction effectiveness, and early termination. Then, the cost of CART is bounded by $\mathcal{O}(\tilde{n}d)$, BR is bounded by $\mathcal{O}(\tilde{n}^d)$, and MHABR is bounded by $\mathcal{O}(\tilde{n}^2 \frac{d(d-1)}{2})$ for $d \geq 2$.*

Proof. Given a maximum of $\tilde{n}$ split candidates at each layer, the greedy CART algorithm determines tree parameters node by node, as detailed in Section 8.1. Consequently, the computational cost of CART at each layer is independent of other layers, and the aggregate computational complexity can be expressed as $\mathcal{O}(\tilde{n}d)$.

Subsequently, the BR method, being a global optimization approach, recursively solves for the subtrees until a depth of 1 is reached. Therefore, its computational cost is determined by the product of the cost at each of the d layers, yielding a complexity of $\mathcal{O}(\tilde{n}) \times \cdots \times \mathcal{O}(\tilde{n}) = \mathcal{O}(\tilde{n}^d)$.

Then, based on the costs of CART and BR, we can obtain the cost of ABR. For the *LLP*, there are left and right subtrees solved by CART. We suppose that at most $\tilde{n}_L$ and $\tilde{n}_R$ split candidates for the left and right subtrees, respectively. Then we have $\tilde{n}_L + \tilde{n}_R \leq \tilde{n}$. The subtrees have depth $d - 1$, the cost is $\mathcal{O}(\tilde{n}_L(d-1)) + \mathcal{O}(\tilde{n}_R(d-1)) \leq \mathcal{O}(\tilde{n}(d-1))$. Then for the *ULP*, the cost is still bounded by $\tilde{n}$. So, the total cost of ABR is bounded by $\mathcal{O}(\tilde{n}^2(d-1))$ for $d \geq 2$.

At last, we consider the cost of MH. For each MH iteration with the depth d_{sub}, the cost is $\mathcal{O}(\tilde{n}^2(d_{sub}-1))$, and MH runs at most $2^{d-1}-1$ iterations. So the cost of `MHABR` is calculated as

$$\begin{aligned}
&\tilde{n}^2(d-1) + \sum_{i=2}^{2} \tilde{n}_i^2(d-2) + \cdots + \sum_{i=2^{d-2}}^{2^{d-1}} \tilde{n}_i^2 \\
&\leq \tilde{n}^2(d-1) + (\sum_{i=1}^{2} \tilde{n}_i)^2(d-2) + \cdots + (\sum_{i=2^{d-2}}^{2^{d-1}} \tilde{n}_i)^2 \\
&\leq \tilde{n}^2(d-1) + \tilde{n}^2(d-2) + \cdots + \tilde{n}^2 \\
&= \tilde{n}^2 \frac{d^2-d}{2}
\end{aligned} \tag{23}$$

Therefore, we can conclude that the cost of `MHABR` is bounded by $\mathcal{O}(\tilde{n}^2 \frac{d(d-1)}{2})$ for $d \geq 2$. Note that, when $d = 1$, all the methods are bounded by $\mathcal{O}(\tilde{n})$. □

These bounds are intended only as coarse worst-case estimates; in practice, the observed runtime is substantially reduced by warm-starting, reduction, and early stopping.

5 Numerical Experiments

This section presents a comprehensive empirical evaluation of our proposed algorithm across various benchmark datasets (detailed in Section 9). We assess key performance metrics, specifically predictive accuracy and computational efficiency, against five primary baseline methods. These baselines encompass widely used heuristics, including `CART` (Sadeghi et al., 2022), `LS-OCT` (Dunn, 2018), and `DPDT` (Kohler et al., 2025), as well as state-of-the-art global optimization solvers such as `DL8.5` (Aglin et al., 2020) and `Quant-BnB` (Mazumder et al., 2022). Furthermore, due to the lack of an open-source implementation for direct reproducibility, we provide a supplementary comparison against the self-reported results of `TAO` (Carreira-Perpinán & Tavallali, 2018) in Section 7. A central focus of this evaluation is demonstrating three key advantages of our method: (i) it consistently recovers the global optimum at depth $d = 2$; (ii) it achieves vast computational efficiency gains over exact global solvers at depths $d > 2$ with only a marginal trade-off in optimality; and (iii) it can well scale to deep trees while delivering superior accuracy compared to heuristic baselines.

Datasets and computing environment: We collected 57 classification datasets from the UCI Machine Learning Repository (Dua & Graff, 2017), spanning both binary and multi-class tasks, with sample sizes ranging from 47 to 60,807,600. The datasets were categorized into 3 groups: 51 small-scale datasets ($n < 10K$) and 5 medium-scale ($10K \leq n \leq 1M$), and 3 large-scale datasets ($n \geq 1M$). Before the experiments, each dataset was randomly split, with 75% for training and 25% for testing, respectively. The results reflect the average of 10 runs for each dataset. All experiments are conducted on a 40-core *Intel Xeon Gold 5115 CPU* (2.40 `GHz`) with 93.9 `GB RAM`.

Implementation: Our algorithm is implemented in `Julia`, with the source code publicly available on GitHub[1]. We also provide an enhanced version of `CART` (Section 8.2), which is developed from `DecisionTree.jl` package and integrated in `MHABR`. All the baselines are obtained from their official repository, except `LS-OCT`, which we reproduced in `Julia`. Time limits are set to 4 hours for small-scale datasets and 24 hours for larger datasets. More details are given in Section 9.3.

5.1 Performance on Small Datasets

We evaluate our algorithm on 51 small datasets, encompassing both an aggregate analysis to illustrate the relative performance of the methods and detailed individual comparisons. Since global methods often suffer from limited scalability while heuristic approaches frequently compromise on optimality, we defer a granular analysis of these trade-offs to the scenario-specific discussions.

[1] `https://anonymous.4open.science/r/Anonymous_TMLR-B68D`

Table 1 provides an overall results of these methods. To rigorously assess optimality, we also utilize binarized datasets to benchmark the optimality gap against DL8.5. Among the competing methods, DPDT emerges as the strongest baseline; like our approach, it is non-greedy, yet it offers superior efficiency and scalability compared to other baselines. Furthermore, DPDT can be configured to mimic our root-node search strategy by adjusting the (N,p) parameters. Consequently, to ensure a comprehensive comparison, we explicitly evaluate this variant, denoted as DPDT(Np).

Table 1: Average training and testing accuracy on 51 small datasets

	Depth	Binarized Datasets		Original Datasets (ODs)					
		DL8.5	MHABR$_{bin}$	Quant-BnB	LS-OCT	CART	DPDT	DPDT(Np)	MHABR
Training accuracy (%)	2	83.18*	83.18	**84.31***	83.90	81.84	83.54	83.78	**84.31***
	3	87.71*	86.97	**89.44***	87.59	85.96	88.19	88.02	88.93
	4	90.79*	89.89	/	90.14	89.02	90.95	91.00	**92.26**
	8	95.89*	94.79	/	96.94	96.70	97.72	98.01	**98.55**
Testing accuracy (%)	2	79.18	75.14	79.89	79.67	78.63	79.52	79.64	**79.91**
	3	81.56	79.82	82.56	82.18	81.13	81.96	82.08	**82.42**
	4	82.42	81.52	/	**83.53**	82.67	83.30	83.16	83.30
	8	82.62	83.58	/	85.20	85.20	**85.35**	84.95	84.10

[1] "*" denotes global optimal solutions. [2] **Bold** numbers indicate the best result. [3] *bin.* denotes binarized datasets.
[4] Underlined numbers indicate the average over 42 datasets that Quant-BnB completes. [5] Quant-BnB fails to converge for depths 4 and 8.

Accuracy: As shown in Table 1, MHABR achieves the strongest average *testing* accuracy among the compared methods at depths $d = 2$ and $d = 3$ (excluding Quant-BnB on datasets for which it does not converge). At $d = 8$, the gap between training and testing accuracy indicates some overfitting on the small datasets, so these results should be interpreted together with validation-based tuning results reported later. Even in this regime, however, MHABR maintains an average **0.99%** advantage over DL8.5. A broader comparison with non-global baselines on medium- and large-scale datasets, where overfitting is less pronounced, is provided in the following subsection.

In terms of training accuracy, MHABR consistently achieves the highest values among the heuristic baselines, outperforming CART by an average of **2.63%**. Since training accuracy more directly reflects the quality of the optimized tree on the training objective, we use it here as an indicator of optimization quality rather than generalization performance. Under this metric, Quant-BnB is competitive only at $d = 3$, where it completes 42 datasets and yields an average gap of 0.51% relative to MHABR, but it fails to converge for deeper trees. On binarized datasets, MHABR$_{\text{bin.}}$ performs slightly below DL8.5, with an average gap of 0.69%, whereas standard MHABR on the original continuous datasets exceeds DL8.5 by an average of 1.62%. Notably, MHABR attains the **global optimum** at $d = 2$ on both dataset types. Although DPDT(Np) slightly improves upon standard DPDT in training accuracy, it remains on average 0.81% below MHABR, suggesting that the MH refinements improve optimization quality beyond the one-shot approximate search.

Scalability and efficiency: A key property of MHABR is its scalability, demonstrated by a slower increase in computational cost with tree depth compared to global optimal methods, while maintaining higher accuracy than heuristic baselines. As shown in Figure 3, across the 51 small-scale datasets, Quant-BnB becomes prohibitively expensive for $d \geq 3$, and DL8.5 exhibits a sharp rise in computational cost with increasing depth, particularly between depths 4 and 8, eventually exceeding the runtime of MHABR. In contrast, MHABR exhibits nearly linear growth in computational cost, similar to other heuristic methods, which highlights its superior scalability for deeper trees.

5.2 Medium-scale and Large-scale Datasets

This section evaluates the proposed method on the challenge of learning deep trees (for $d = 4, 8$) for large-scale datasets, where Quant-BnB fails for all cases and DL8.5 only succeeds on several datasets on depth 4 and fails on depth 8. Table 2 shows results using five medium and three large datasets compared to 3 baselines.

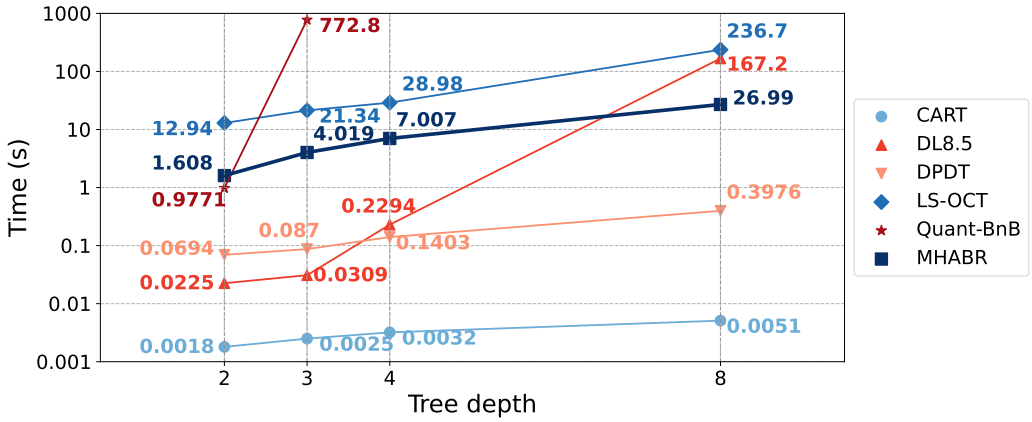


Figure 3: Training time on 51 small datasets.

For large-scale applications of our algorithm, we incorporate additional techniques (detailed in Section 9.2) to improve the efficiency of our algorithm, as detailed in the following, which is further analyzed in Section 5.6.

Techniques for large-scale datasets: mini-batch and tolerance termination To enhance efficiency for large-scale datasets, mini-batch sampling and an ε-tolerance termination criterion are employed. Mini-batch sampling uses a subset of the dataset, defined by a sampling ratio $\theta \in (0, 1]$, as input to `ABR`, while the ε-tolerance criterion determines termination. Although this approach may slightly reduce accuracy, it significantly decreases computational time, as demonstrated later. The primary factor affecting computational time is the number of splits in the initial branch, influenced by dataset characteristics, with continuous features often contributing more distinct values, many of which minimally impact the loss function.

For the mini-batch strategy, a proportion θ of the original data is sampled for each subtree. On large-scale datasets, continuous features often require fewer samples, as their splits generally result in only small fluctuations in the overall loss. To further enhance the efficiency of this algorithm for extremely large-scale datasets, we introduce a tolerance parameter ε as a termination criterion. Let $\texttt{length}(\mathbb{K})$ denote the size of the split index set. The termination criterion can be formally defined as:

$$\texttt{length}(\mathbb{K})/n \leq \varepsilon. \tag{24}$$

For a fixed feature index a, the candidate set of the `BR` method, denoted by $\mathcal{K}^a$, contains n_a elements. If no element is reduced during the process, the bisection branching strategy generates a set $\mathbb{K}$ with at most $n_a/2^j$ elements after the j-th iteration. By Equation (24), the algorithm will terminate after at most $\lceil \log \frac{n_a}{n\varepsilon} \rceil$ iterations. Since the algorithm generates two subsets per iteration, the total number of splits evaluated throughout this process is $\texttt{length}(\mathbb{K}) \leq \varepsilon \cdot n$ when the termination criterion is satisfied.

Performance on medium and large datasets: Across the evaluated datasets and baselines, `MHABR` achieves the the highest average test accuracy among the compared methods on these datasets, while remaining computationally feasible for deeper trees. On average, it outperforms `CART` by **3.66%** and `DPDT` by **3.01%** in testing accuracy at $d = 8$. On medium datasets, although solvers like `DPDT` often require less runtime, `MHABR` uses additional computation to achieve higher test accuracy on these datasets. Specifically, across the five medium-scale datasets, `MHABR` significantly improves testing accuracy, outperforming `CART` by an average of 1.84% at depth 4 and 4.59% at depth 8, while exceeding `DPDT` by 0.66% at depth 4 and 3.49% at depth 8. The largest improvement is observed on the *Avila* dataset at $d = 8$, where `MHABR` surpasses all other methods by at least 13.35% in training accuracy and 14.53% in testing accuracy. Furthermore, on large-scale datasets where exact methods like `LS-OCT` fail entirely, `MHABR` achieves the highest test accuracy among the compared methods on these datasets. It consistently converges alongside `CART` and `DPDT`, outperforming `CART` by an average of 1.14% at depth 4 and 4.13% at depth 8, while exceeding `DPDT` by 0.63% at depth 4 and 3.89% at depth 8.

Table 2: Performance comparison on 5 medium-scale (top) and 3 large-scale datasets (bottom).

Dataset	n	p	n_c	Method	$d = 4$			$d = 8$		
					Train (%)	Test (%)	Time (s)	Train (%)	Test (%)	Time (s)
Avila	10,430	10	12	CART	57.24	56.58	0.03	76.77	74.87	0.05
				DPDT	58.71	58.85	0.90	79.71	77.00	2.96
				LS-OCT	60.33	59.87	183.61	79.84	77.00	1585.04
				MHABR	**61.59**	**60.75**	8.42	**93.19**	**91.53**	14.21
Eeg	14,980	14	2	CART	70.12	69.30	0.03	79.31	76.30	0.06
				DPDT	72.86	71.92	1.20	83.72	79.48	4.24
				LS-OCT	71.94	71.16	326.68	81.93	78.43	2498.00
				MHABR	**74.50**	**72.84**	23.98	**88.93**	**82.51**	102.00
Htru	17,898	8	2	CART	97.94	97.83	0.05	98.66	**97.75**	0.08
				DPDT	98.14	**97.92**	1.63	98.87	97.68	4.82
				LS-OCT	98.11	**97.92**	230.05	98.74	97.66	1897.76
				MHABR	**98.23**	97.81	521.25	**99.26**	97.42	3363.61
Shuttle	43,500	9	7	CART	99.83	99.80	0.03	**100.00**	99.95	0.04
				DPDT	99.90	99.88	0.93	**100.00**	**99.96**	1.64
				LS-OCT	99.92	99.89	665.52	**100.00**	**99.96**	5655.26
				MHABR	**99.97**	**99.95**	28.53	**100.00**	99.95	30.09
Skin-seg.	245,057	3	2	CART	97.42	97.37	0.08	99.50	99.49	0.10
				DPDT	98.22	98.20	5.48	99.76	99.74	10.58
				LS-OCT	98.41	98.39	1471.57	99.75	99.73	15259.16
				MHABR	**98.72**	**98.72**	46.62	**99.95**	**99.91**	164.83
SUSY	5,000,000	18	2	CART	76.60	76.61	29.32	78.38	78.34	75.79
				DPDT	76.86	76.87	1197.44	78.72	78.68	2951.08
				MHABR	**77.84**	**77.86**	17632.26	**78.94**	**78.89**	11953.41
HIGGS	11,000,000	28	2	CART	65.58	65.55	81.98	69.47	69.40	216.10
				DPDT	66.20	66.18	3223.50	69.67	69.57	7158.52
				MHABR	**66.88**	**66.85**	11826.94	**70.17**	**70.05**	78370.11
WESAD	60,807,600	8	8	CART	57.46	57.46	123.23	80.27	80.26	293.24
				DPDT	61.21	61.20	5225.47	79.45	79.45	14623.90
				MHABR	**61.33**	**61.33**	1464.82	**85.37**	**85.38**	7977.35

[1] **Bold** numbers indicate the best result among all the methods. [2] LS-OCT fails to converge on large datasets. [3] MHABR is a light version of MHABR incorporating additional techniques.

5.3 Compared to DPDT

DPDT is one of the strongest heuristic baselines in our experiments. In this subsection, we provide a more detailed comparison. We consider the DPDT with different parameter configuration and α-tuning to show MHABR achieves better optimization quality while remaining efficient with theoretically same searching space.

Recall the training results of $d = 8$ across 51 datasets in Table 1; more specifically, MHABR strictly outperforms DPDT on 23 datasets with an overall improvement of 46.29%, while being only 3.93% worse on 7 datasets. These results demonstrate that MHABR achieves better optimization quality compared to default DPDT.

For a detailed comparison with DPDT under comparable settings, we also compare to DPDT(NP), cart_nodes_list=(Np,), which enables it to achieve the same theoretical accuracy as ABR. To illustrate the performance gains brought by the reduction and MH techniques, we use the datasets Avila and Eeg to plot the corresponding Pareto fronts for CART, DPDT, DPDT(NP), and MHABR. We evaluate tree depths from 2 to 20. For MHABR, we report results only up to $d = 15$, since it already reaches almost 100% training accuracy at depth 10 and begins to show signs of overfitting thereafter.

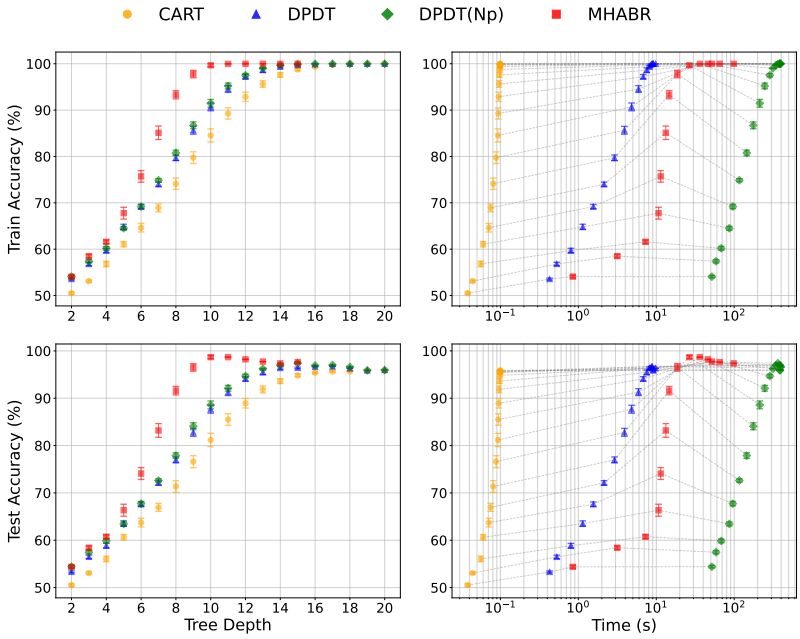


Figure 4: Training/testing accuracy vs. tree depth/running time on `Avila`

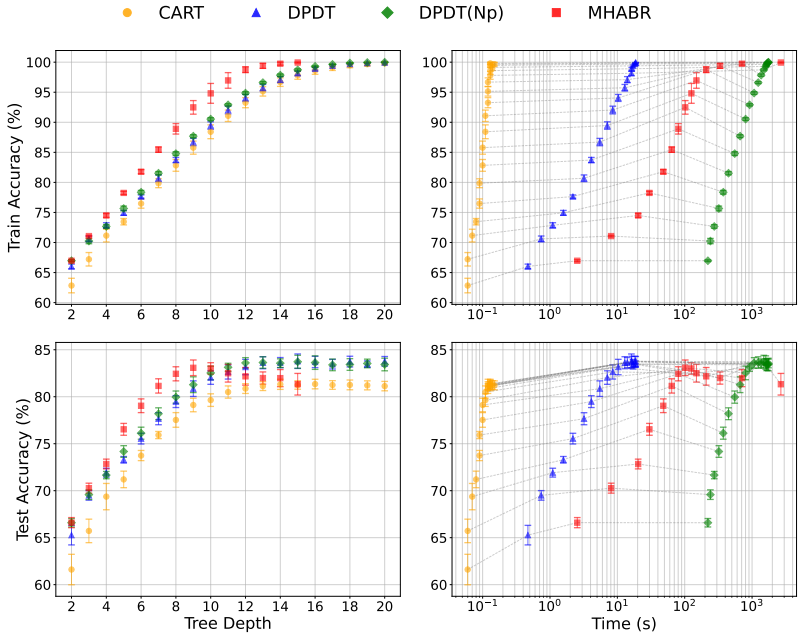


Figure 5: Training/testing accuracy vs. tree depth/running time on `Eeg`

The Pareto fronts shown in Figures 4 and 5 illustrate the trade-offs between training/testing accuracy and tree depth, as well as between training/testing accuracy and training time. To clearly show the relationship between tree depth and running time, we also connect the points of each same depth in the figures of training/testing accuracy vs. time. In Figure 4, among all methods, `MHABR` achieves the highest training accuracy and the highest peak test accuracy among the compared methods in this experiment. It achieves almost perfect training accuracy 99.66% at depth $d = 10$, while its testing accuracy also achieves the highest peak performance 98.68%. Beyond this regime, its testing accuracy declines, reflecting its tendency to overfit

at larger depths. In contrast, DPDT and DPDT(Np) approach this level at least $d \geq 14$, CART at least $d \geq 17$. In Figure 5, MHABR shows weaker testing accuracy when $d \geq 10$, which is expected due to overfitting, as its training accuracy is nearly 100%. This leads to a drop in testing accuracy as well as the running time increases. Nonetheless, excluding overfitting, MHABR still outperforms the other methods.

In terms of running time, CART is the fastest method, but it requires substantially deeper trees (i.e., much larger models) to achieve competitive accuracy. DPDT improves accuracy relative to CART with only a modest increase in computation, but its strength is mainly evident in its default configuration. For achieving higher accuracy, MHABR performs better and requires less time than DPDT(Np). Since model size is crucial for the interpretability of decision trees, obtaining high accuracy with a smaller model is particularly desirable, which is an advantage offered by MHABR.

MHABR achieves higher accuracy than DPDT(Np) at the same depth (before overfitting), demonstrating the substantial performance gains contributed by the MH component. Moreover, MHABR requires less training time than DPDT(Np) before overfitting occurs, highlighting its superior efficiency enabled by the Reduction technique. Although the slope of the running-time increase for MHABR becomes steeper than that of DPDT(Np) once the depth exceeds 13, an effect attributable to the MH procedure, consistent with our complexity analysis, the training accuracy has already reached 100% at this point and cannot improve further. Thus, MH iterations can be relaxed or omitted beyond this depth.

Table 3: α-tuning results of 25 datasets with $n \geq 500$.

Depth	Training Accuracy				Testing Accuracy			
	CART	DPDT	DPDT(Np)	MHABR	CART	DPDT	DPDT(Np)	MHABR
2	81.29 (1.19)	82.54 (1.13)	82.57 (1.06)	**82.89** (1.03)	80.07 (1.94)	81.05 (1.87)	81.09 (1.93)	**81.34** (3.16)
3	84.29 (1.49)	85.69 (1.33)	85.78 (1.19)	**86.54** (1.12)	82.57 (1.85)	83.47 (1.61)	83.54 (1.71)	**83.92** (1.77)
4	86.66 (1.44)	87.93 (1.56)	88.23 (1.23)	**89.20** (1.53)	84.43 (1.79)	84.87 (1.69)	84.77 (1.76)	**85.40** (1.90)
5	88.35 (1.91)	90.27 (2.12)	90.47 (1.90)	**91.76** (1.77)	85.42 (2.02)	86.11 (2.05)	86.26 (2.12)	**86.31** (1.95)

α-tuning To better assess performance in a realistic setting, we split the data into 50% for training, 25% for validation, and 25% for testing to evaluate the effect of α-tuning (with $\alpha \in [0.0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2]$). To provide a fairer evaluation avoiding the influence of overfitting, we consider 25 datasets that are larger than 500, across depths 2–5. As shown in Table 3, MHABR consistently outperforms the other methods in training and testing accuracy across all depths.

5.4 Discussion about comparison with global optimal methods

In this subsection, we provide a comparative analysis of MHABR and global optimal methods: DL8.5 and Quant-BnB,, focusing on small datasets for which all evaluated methods successfully identified feasible solutions. The results show that our method achieves higher training accuracy than DL8.5 in this setting and offers a favorable empirical trade-off between optimization quality and runtime relative to Quant-BnB.

We use box plots, as shown in Figure 6, to illustrate the results across all datasets. Although DL8.5 is a globally optimal method, it relies on the binarization of continuous datasets. This preprocessing step introduces approximation errors, leading to worse training accuracy compared to the other two methods. This negative impact is particularly evident in the training results at depth 8. In contrast, MHABR achieves higher training accuracy than DL8.5 on these datasets, especially at depth 8. Meanwhile, Quant-BnB can handle the original continuous datasets natively but is limited to a maximum depth of 3 (successfully completing only 42 datasets). Table 4 presents the results for these 42 datasets at depths 2 and 3. MHABR achieves training accuracy equivalent to that of Quant-BnB at $d = 2$. At $d = 3$, it demonstrates only a minor disparity, performing marginally lower than Quant-BnB. This outcome suggests that the solutions generated by MHABR closely approximate the global optimal solutions for shallow trees. Furthermore, it indicates that the potential loss of optimality resulting from the *LLP* approximation is effectively compensated for by the MH procedure.

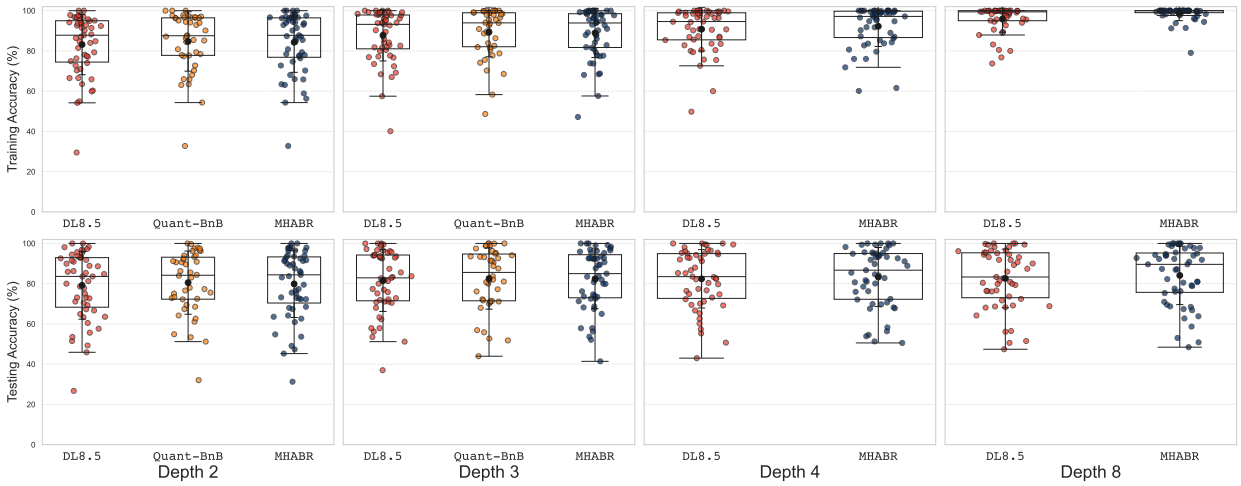


Figure 6: Training and testing accuracy of 51 small datasets across $d = 2, 3, 4, 8$.

Table 4: Comparison on 42 small datasets (excluding 9 datasets `Quant-BnB` that fail to complete).

	Depth	`CART`	`DL8.5`	`Quant-BnB`	`MHABR`
Train (%)	$d = 2$	84.10 ± 14.11	$83.93 \pm 15.04^*$	$\mathbf{84.70 \pm 14.76}^*$	$\mathbf{84.70 \pm 14.76}^*$
	$d = 3$	87.99 ± 12.41	$88.08 \pm 13.18^*$	$\mathbf{89.44 \pm 12.36}^*$	88.86 ± 12.62
Test (%)	$d = 2$	78.98 ± 16.37	79.89 ± 16.40	$\mathbf{80.53 \pm 15.71}$	80.47 ± 15.91
	$d = 3$	81.41 ± 14.97	81.71 ± 15.79	$\mathbf{82.56 \pm 15.22}$	82.30 ± 15.14

[1] **Bold** numbers indicate the best solutions. [2] "$*$" signifies the global optimum in the corresponding formulation.
[3] Even though `Quant-BnB` and `MHABR` can obtain the optimal loss when $d = 2$, the optimal trees might differ.

5.5 Optimality Gap Evaluation and An Extension

To assess the optimality of our algorithm, we evaluate 10 datasets (from the 51 small datasets) and compute the optimality gap as the relative difference between the solution produced by `MHABR` and the global optimum obtained by `BR`. We further introduce an extended variant, `MHABR`^{2} (2-step lookahead method), which reduces the optimality gap by extending the *ULP* from the root node to the top two layers (nodes $\{1, 2, 3\}$) for $d \geq 3$. Importantly, `MHABR`^{2} attains exact optimal solutions for depth-3 trees and further reduces the optimality gap for deeper trees.

`MHABR`^{2}: Our algorithm is implemented in `Julia`. For the default version, we adopt the reduction strategy with parameters $\varepsilon = 0$ and $\theta = 1$. To construct `MHABR`^{2}, which enlarges the *ULP* to include the top two layers of nodes, we modify the evaluation function in `ABR` recursion as follows:

$$\texttt{MHABR}: \begin{cases} \texttt{BR}, & d - d_{sub} \leq 1 \\ \texttt{CART}, & \text{otherwise} \end{cases} \quad \Rightarrow \quad \texttt{MHABR}^2: \begin{cases} \texttt{BR}, & d - d_{sub} \leq 2 \\ \texttt{CART}, & \text{otherwise.} \end{cases}$$

First, we illustrate the scope of our method in Figure 7, which summarizes the theoretical relationships among the proposed approaches. As noted in Theorem 4.1, our algorithm guarantees solutions no worse than `CART` when no reductions are applied, with the MH procedure further refining results toward better optimality. Expanding the `ULP` scope improves the approximation, making `ABR`^{2} superior to `ABR`, and further enlargement brings methods closer to global optimal approaches such as the unapproximated `BR` and `Quant-BnB`. For binarized datasets, the binarization process introduces errors that reduce optimality, so `MHABR`$_{\text{bin.}}$ performs worse than the original dataset version and is clearly inferior to global optimal methods such as `DL8.5` and `MurTree`.

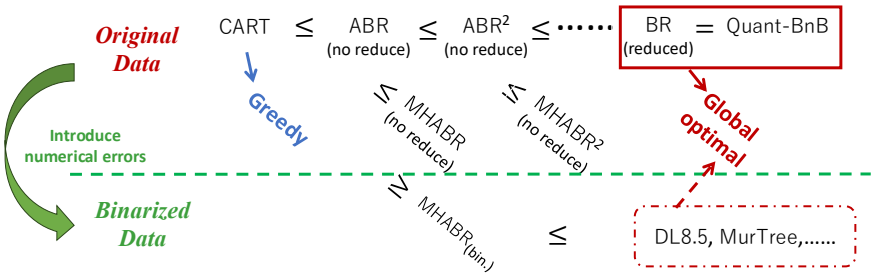


Figure 7: Theoretical relationships in terms of optimality between our algorithms and other global optimal methods

We evaluate the optimality gap using 10 small datasets solvable by `BR`, with results reported in Tables 5 and 6. As shown, `CART` exhibits a relatively large gap, typically exceeding 3% on average at both depths 3 and 4. In contrast, `MHABR` maintains a gap within 1% on average, while the extended variant, `MHABR`^{2}, further improves optimality, consistently reducing the gap below 1%. Correspondingly, the computational cost increases from `CART` to `BR` as optimality improves. Therefore, we generally recommend `MHABR`, which offers strong optimality while keeping the computational cost reasonable.

Table 5: Gap to optimal of `CART`, `MHABR`, `MHABR`^{2}, and `BR` across 10 datasets ($d = 3$)

Dataset	n	p	n_c	`CART` (greedy)		`MHABR`		`MHABR`^{2} (optimal)		`BR` (optimal)	
				Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
Iris	150	4	3	2.68	0.0004	0.18	0.02	0.00	0.09	0.00	0.09
Haberman-survival	306	3	2	3.97	0.0002	0.69	0.01	0.00	0.12	0.00	0.12
Monks-problems-3	554	6	2	2.41	0.0002	2.46	0.01	0.00	0.02	0.00	0.02
Monks-problems-1	556	6	2	12.15	0.0005	1.02	0.01	0.00	0.02	0.00	0.02
Monks-problems-2	600	6	2	2.70	0.0019	0.45	0.01	0.00	0.02	0.00	0.02
Balance-scale	625	4	3	3.45	0.0001	1.09	0.01	0.00	0.02	0.00	0.02
Blood-transfusion	748	4	2	2.36	0.0001	1.03	0.02	0.00	0.45	0.00	0.45
Mammographic-mass	830	5	2	1.16	0.0004	0.22	0.02	0.00	0.31	0.00	0.31
Contraceptive-method-choice	1,473	9	3	9.95	0.0010	1.00	0.03	0.00	0.51	0.00	0.51
Car-evaluation	1,728	6	4	3.72	0.0001	0.60	0.01	0.00	0.07	0.00	0.07

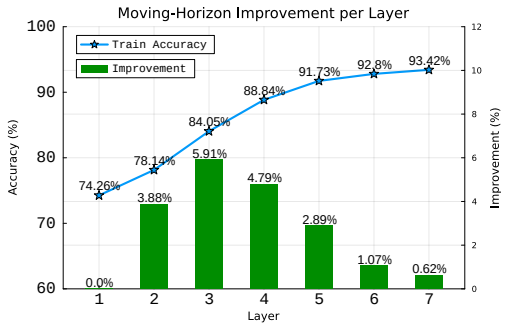
Table 6: Gap to optimal of `CART`, `MHABR`, `MHABR`^{2}, and `BR` across 10 datasets ($d = 4$)

Dataset	n	p	n_c	`CART` (greedy)		`MHABR`		`MHABR`^{2}		`BR` (optimal)	
				Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
Iris	150	4	3	0.54	0.0005	0.00	0.02	0.00	0.30	0.00	8.36
Haberman-survival	306	3	2	6.94	0.0002	1.71	0.03	0.20	0.24	0.00	5.05
Monks-problems-3	554	6	2	0.00	0.0002	0.00	0.02	0.00	0.04	0.00	0.24
Monks-problems-1	556	6	2	16.26	0.0006	2.48	0.02	0.07	0.04	0.00	0.25
Monks-problems-2	600	6	2	4.46	0.0021	1.13	0.03	0.64	0.05	0.00	0.27
Balance-scale	625	4	3	5.66	0.0001	0.78	0.03	0.40	0.06	0.00	0.39
Blood-transfusion	748	4	2	4.00	0.0001	1.23	0.08	0.49	1.08	0.00	34.71
Mammographic-mass	830	5	2	2.36	0.0005	0.62	0.07	0.22	0.73	0.00	15.38
Contraceptive-method-choice	1,473	9	3	5.76	0.0011	1.19	0.11	0.27	1.40	0.00	28.79
Car-evaluation	1,728	6	4	3.82	0.0001	0.11	0.05	0.00	0.15	0.00	1.24

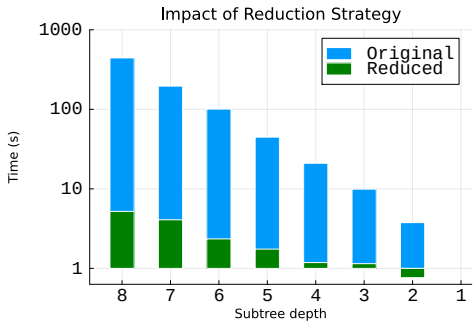
5.6 Ablation Studies

We now study how the reduction strategy, the MH method, and tunable parameters ε, θ affect the computational efficiency and accuracy of `MHABR`, by systematically evaluating MH procedure performance per layer using an `Avila` case study on a depth-8 decision tree.

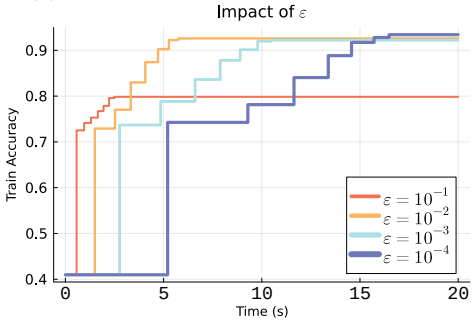
To evaluate the contribution of the MH refinement, we show the results of a depth-8 tree on the Avila dataset, without any other techniques except the approximation. The result is shown in Figure 8(a). We observe that



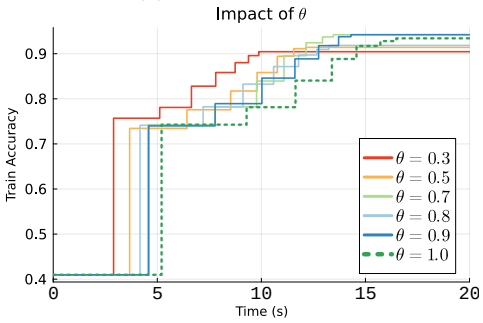
(a) Impact of MH on training accuracy



(b) Reduction strategy



(c) Terminate tolerance



(d) Mini-batch

Figure 8: Impact of the MH procedure, reduction strategy, and parameters ε and θ. We train a depth-8 tree for `Avila`, where (c) and (d) systematically evaluate MH performance for different subtree depths. For instance, the depth-8 subtree represents the complete original tree, while the depth-7 subtrees are rooted at nodes 2 and 3. Similarly, the depth-6 subtrees are rooted at nodes 4, 5, 6, and 7, and this pattern continues for the other depths.

the most significant improvement occurs within the first five layers. This suggests that the MH refinements have a greater impact on earlier branch nodes, where the corresponding subtrees are relatively larger and contain more samples.

Figure 8(b) demonstrates that the reduction strategy significantly accelerates the computation of subtrees at various depths within the MH procedure. It is evident that the reduction strategy yields a decrease in runtime of nearly two orders of magnitude for the depth-8 subtree. While this reduction in computational time becomes less pronounced as the subtree depth decreases, it still contributes to a considerable overall cost reduction.

Figure 8(c) illustrates the convergence of results obtained through iterative refinement using the MH under varying termination conditions. The final accuracies achieved with $\varepsilon = \{10^{-2}, 10^{-3}, 10^{-4}\}$ are notably similar, but $\varepsilon = 10^{-2}$ results in 60% reduction in running time. The scenario with $\varepsilon = 10^{-1}$ exhibits a discernible decrease in accuracy. Overall, larger values of ε lead to lower accuracy for `MHABR`, but with the benefit of decreased training time. This suggests the possibility of identifying a suitable ε that incurs a minor reduction in accuracy while yielding a significant saving in computational time, as exemplified by $\varepsilon = 10^{-2}$ in this case.

The influence of batch sizes is explored in Figure 8(d). Utilizing mini-batches involves training a model on a subset of the data, which invariably reduces computational time. Interestingly, this approach can sometimes yield superior performance. As depicted in the figure, the accuracies achieved with batch size ratios (θ) of 0.8 and 0.9 surpass those obtained when using the complete dataset. Conversely, a batch size ratio of $\theta = 0.3$ results in a relatively discernible reduction in accuracy (but less than 5%).

6 Conclusions and Discussions

We introduced `MHABR`, an approximate method for training deep classification trees within a bilevel optimization framework. The method combines a root-level branch-and-reduce search, an approximate lower-level solver based on CART, and a moving-horizon refinement procedure. The resulting algorithm is exact for depth-2 trees and performs well empirically on the benchmark datasets considered in this paper. In particular, it achieves strong test accuracy relative to the compared baselines while remaining computationally feasible on larger datasets than the exact global methods included in our study. These results suggest that `MHABR` offers a favorable empirical trade-off between optimization quality, runtime, and scalability.

References

Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Strong optimal classification trees. *Operations Research*, 2024.

Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. 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.

Josep Alòs, Carlos Ansótegui, and Eduard Torres. Interpretable decision trees through maxsat. *Artificial Intelligence Review*, 56(8):8303–8323, 2023.

Florent Avellaneda. Efficient inference of optimal decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3195–3202, 2020.

Dimitri Bertsekas. *A course in reinforcement learning*. Athena Scientific, 2024.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106:1039–1082, 2017.

Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Cart. *Classification and Regression Trees*, 1984.

Miguel A Carreira-Perpinán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. *Advances in neural information processing systems*, 31, 2018.

Miguel Á Carreira-Perpiñán and Arman Zharmagambetov. Ensembles of bagged tao trees consistently improve over random forests, adaboost and gradient boosting. In *Proceedings of the 2020 ACM-IMS on foundations of data science conference*, pp. 35–46, 2020.

Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

Joseph Dunn. *Optimal Trees for Prediction and Prescription*. Ph.d. thesis, Massachusetts Institute of Technology, 2018.

Kenneth Dwyer and Robert Holte. Decision tree instability and active learning. In *European conference on machine learning*, pp. 128–139. Springer, 2007.

Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, 2014.

Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, 81:233–260, 2021.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. 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.

Kaixun Hua, Jiayang Ren, and Yankai Cao. A scalable deterministic global optimization algorithm for training optimal decision tree. *Advances in Neural Information Processing Systems*, 35:8347–8359, 2022.

Tim Huisman, Jacobus GM van der Linden, and Emir Demirović. Optimal survival trees: A dynamic programming approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 12680–12688, 2024.

Hector Kohler, Riad Akrour, and Philippe Preux. Breiman meets bellman: Non-greedy decision trees with mdps. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 1207–1218, 2025.

Martin Krzywinski and Naomi Altman. Classification and regression trees. *Nature Methods*, 14(8):757–758, 2017.

Hyafil Laurent and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pp. 6150–6160. PMLR, 2020.

Rahul Mazumder, Xiang Meng, and Haoyue Wang. Quant-bnb: A scalable branch-and-bound method for optimal decision trees with continuous features. In *International Conference on Machine Learning*, pp. 15255–15277. PMLR, 2022.

Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo Seltzer. Fast sparse decision tree optimization via reference ensembles. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 9604–9613, 2022.

Siegfried Nijssen and Elisa Fromont. 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.

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

J. Ross Quinlan. *C4. 5: programs for machine learning*. The Morgan Kaufmann Series in Machine Learning, 1993.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Cynthia Rudin. Why black box machine learning should be avoided for high-stakes decisions, in brief. *Nature Reviews Methods Primers*, 2(1):81, 2022.

Hong S Ryoo and Nikolaos V Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–138, 1996.

Ben Sadeghi, Poom Chiarawongse, Kevin Squire, Daniel C. Jones, Andreas Noack, Cédric St-Jean, Rik Huijzer, Roland Schätzle, Ian Butterworth, Yu-Fong Peng, and Anthony Blaom. DecisionTree.jl - A Julia implementation of the CART Decision Tree and Random Forest algorithms, November 2022. URL `https://doi.org/10.5281/zenodo.7359268`.

William D Shannon and David Banks. Combining classification trees using mle. *Statistics in medicine*, 18(6): 727–740, 1999.

Pouya Shati, Eldan Cohen, and Sheila A McIlraith. Sat-based optimal classification trees for non-binary data. *Constraints*, 28(2):166–202, 2023.

Mohit Tawarmalani and Nikolaos V Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.

Jacobus van der Linden, Mathijs de Weerdt, and Emir Demirović. Necessary and sufficient conditions for optimal decision trees using dynamic programming. *Advances in Neural Information Processing Systems*, 36:9173–9212, 2023.

Jacobus G. M. Van der Linden, Mathijs De Weerdt, and Emir Demirović. Fair and optimal decision trees: A dynamic programming approach. *Advances in Neural Information Processing Systems*, 35:38899–38911, 2022.

Sicco Verwer and Yingqian Zhang. 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.

Arman Zharmagambetov, Magzhan Gabidolla, and Miguel Ê Carreira-Perpiñán. Improved boosted regression forests through non-greedy tree optimization. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Bingzhao Zhu and Mahsa Shoaran. Tree in tree: from decision trees to decision graphs. *Advances in Neural Information Processing Systems*, 34:13707–13718, 2021.

Haoran Zhu, Pavankumar Murali, Dzung Phan, Lam Nguyen, and Jayant Kalagnanam. A scalable mip-based method for learning optimal multivariate decision trees. *Advances in Neural Information Processing Systems*, 33:1771–1781, 2020.

Appendix

7 Supplementary Experiments: Comparison with TAO

We also provide an informal comparison with the well-known heuristic method `TAO` (Carreira-Perpinán & Tavallali, 2018). Because the authors do not provide publicly available code for reproducibility, we evaluated our method on the same five datasets reported in Appendix 2.1 of their paper, using the identical data split (50% training, 25% validation, and 25% testing). The corresponding results are presented below:

Table 7: Performance comparison between `TAO` and `MHABR`. Results are reported as mean accuracy with standard deviation in parentheses. Best results for each dataset, depth, and split are highlighted in bold.

Dataset	**Depth**	**Train**		**Test**	
		`TAO`	`MHABR`	`TAO`	`MHABR`
Balance-scale	$d=2$	72.5 (1.6)	**72.7** (0.7)	**69.5** (2.9)	68.5 (2.0)
	$d=3$	**76.9** (1.1)	76.8 (1.4)	71.6 (1.9)	**74.8** (2.2)
	$d=4$	84.0 (1.5)	**84.1** (1.1)	**79.8** (3.1)	78.9 (2.7)
Banknote-auth.	$d=2$	91.9 (0.4)	**92.8** (0.4)	90.6 (0.9)	**91.3** (1.1)
	$d=3$	96.0 (0.5)	**98.3** (0.4)	95.7 (1.2)	**97.2** (0.6)
	$d=4$	98.9 (0.7)	**99.5** (0.3)	97.2 (0.7)	**98.7** (0.5)
Blood-transfusion	$d=2$	**78.0** (0.8)	77.9 (1.4)	75.8 (2.0)	**77.1** (3.7)
	$d=3$	79.5 (1.0)	**80.3** (1.4)	77.0 (2.0)	**77.8** (2.9)
	$d=4$	**81.6** (1.3)	80.7 (2.3)	77.2 (1.3)	**77.6** (3.5)
Breast-cancer	$d=2$	95.0 (0.5)	**96.4** (0.4)	92.7 (2.2)	**93.9** (1.7)
	$d=3$	97.0 (0.6)	**98.5** (0.5)	93.1 (1.4)	**95.0** (1.7)
	$d=4$	98.0 (0.5)	**99.6** (0.3)	93.2 (0.5)	**95.1** (1.8)
Spambase	$d=2$	86.5 (0.7)	**87.3** (0.5)	86.1 (1.0)	**86.9** (0.6)
	$d=3$	90.0 (0.4)	**90.8** (0.3)	89.1 (1.0)	**90.3** (0.8)
	$d=4$	91.8 (0.3)	**92.3** (0.6)	90.3 (0.8)	**91.3** (0.8)

On average, `MHABR` outperforms `TAO` by 0.7% in training accuracy (88.5% to 87.8%) and 1.0% in testing accuracy (86.3% to 85.3%). `MHABR` consistently outperforms `TAO`, showing at least a 0.7% improvement across the five datasets at depths 2, 3, and 4. Based on this trend, we expect that `MHABR` will demonstrate even greater gains at depth, as the performance gap between the two methods increases with tree depth. Unfortunately, `TAO` does not report results for this setting, and we are unable to reproduce them fairly without access to its implementation.

8 Additional Theoretical Results and Analysis

In this section, we recall key properties of `CART`, provide a detailed introduction to the reduced `CART` algorithm, and discuss the specific scenarios where `MHABR` is guaranteed to outperform this greedy method.

8.1 Recall the basics of `CART`

Before analyzing the ABR method, we recall some basics of `CART`. We first analyze some properties of the DT model. Let $\mathcal{I}_t$ denote the data in the node $t \in \mathcal{N}_B \cup \mathcal{N}_L$, then the potential loss of this node can be calculated as:

$$V_0(\mathcal{I}_t) = \min_{c \in \mathcal{N}_c} L(\mathcal{I}_t, c) \tag{25}$$

The parameters of this node involve minimizing the combined loss of the potential children:

$$a', k' = \arg \min_{a \in [p], k \in \mathcal{K}^a} V_0(\mathcal{I}^a_{t,[0,k]}) + V_0(\mathcal{I}^a_{t,[k,n_a]}) \tag{26}$$

where $\mathcal{I}^a_{t,[0,k]}$ and $\mathcal{I}^a_{t,[k,n_a]}$ respectively represent the data subsets assigned to the left and right child nodes resulting from the split of the data $\mathcal{I}_t$ based on (a, k).

Depth-1 optimality For a sample subset $\mathcal{I} \subseteq [n]$, a depth-1 DT comprises two optimal constant fits, i.e., two optimal depth-0 subtrees, expressed by $V_0(\mathcal{I}) = \min_{c \in \mathcal{N}_c} L(\mathcal{I}, c)$, which denotes the loss of the best constant approximation to $\mathcal{Y}$, in the same manner as `CART`. For $d = 1$, `CART` solves a one-stage decision problem, which, according to the greedy nature, is optimal.

Monotonicity The loss function of `CART` exhibits monotonicity with respect to tree depth, such that $W_d(\mathcal{I}) \leq W_{d-1}(\mathcal{I})$. This is attributed to the monotonic nature of the branching operation concerning the count of correctly classified samples (accuracy). Branching will either increase this count (thereby decreasing the loss) or, in the worst-case scenario where the newly formed leaves offer no improvement over their parent node, the count will remain unchanged.

For example, we consider a node t with data $\mathcal{I}_t$, the potential loss is $V_0(\mathcal{I}_t)$ as defined by Equation (25), with prediction c_1. Then, we branch the node to obtain two branch nodes $\mathcal{I}_{2t}, \mathcal{I}_{2t+1}$, where $\mathcal{I}_{2t} \cup \mathcal{I}_{2t+1} = \mathcal{I}_t$. The losses of these new nodes are $V_0(\mathcal{I}_{2t})$ and $V_0(\mathcal{I}_{2t+1})$ with predictions c_2 and c_3, respectively. Then we have:

$$\begin{aligned} V_0(\mathcal{I}_t) = \sum_{i \in \mathcal{I}} \mathbb{1}\{y_i \neq c_1\} &= \sum_{i \in \mathcal{I}_{2t}} \mathbb{1}\{y_i \neq c_1\} + \sum_{i \in \mathcal{I}_{2t+1}} \mathbb{1}\{y_i \neq c_1\} \\ &\geq \sum_{i \in \mathcal{I}_{2t}} \mathbb{1}\{y_i \neq c_2\} + \sum_{i \in \mathcal{I}_{2t+1}} \mathbb{1}\{y_i \neq c_3\} \\ &= V_0(\mathcal{I}_{2t}) + V_0(\mathcal{I}_{2t+1}), \end{aligned} \tag{27}$$

The inequality holds because c_2 and c_3 are chosen as the predictions that minimize the misclassifications within their respective nodes $2t$ and $2t+1$ (defined as the most frequent class in Equation (25)). By definition, the optimal prediction c_2 for node $\mathcal{I}_{2t}$ must classify at least as many samples correctly within $\mathcal{I}_{2t}$ as any other prediction, including c_1. Thus, $\sum_{i \in \mathcal{I}_{2t}} \mathbb{1}\{y_i \neq c_2\} \leq \sum_{i \in \mathcal{I}_{2t}} \mathbb{1}\{y_i \neq c_1\}$, and similarly for c_3 over $\mathcal{I}_{2t+1}$. Therefore, the loss after branching is less than or equal to the value before branching.

8.2 A reduced `CART` method

Algorithm 5 A reduced `CART`

function `R-CART`$(\mathcal{I})$:
  for $a \in [p]$ **do**
    Sort $\mathcal{I}$ to obtain $\mathcal{I}^a$, $\mathcal{K}^a$, and set $\mathbb{K} \leftarrow \{\mathcal{K}^a\}$;
    for $\bar{k} \in \mathbb{K}$ **do**
      Select $\bar{k} \leftarrow \mathbb{K}[1]$, obtain $\mathcal{I}^a_{[0,\bar{k}]}$, $\mathcal{I}^a_{[\bar{k},n_a]}$; # *select the first element*
      Compute $V_0(\mathcal{I}^a_{[0,\bar{k}]}) \leftarrow \min_c L(\mathcal{I}^a_{[0,\bar{k}]}, c)$, and $c_1 \leftarrow \arg\min_c L(\mathcal{I}^a_{[0,\bar{k}]}, c)$;
      Compute $V_0(\mathcal{I}^a_{[\bar{k},n_a]}) \leftarrow \min_c L(\mathcal{I}^a_{[\bar{k},n_a]}, c)$, and $c_2 \leftarrow \arg\min_c L(\mathcal{I}^a_{[\bar{k},n_a]}, c)$;
      Calculate $V_1(\mathcal{I}, a, \bar{k}) = V_0(\mathcal{I}^a_{[0,\bar{k}]}) + V_0(\mathcal{I}^a_{[\bar{k},n_a]})$;
      if $V_d(\mathcal{I}, a, \bar{k}) < U$ **then**
        Update $U \leftarrow V_d(\mathcal{I}, a, \bar{k})$, $T \leftarrow [a, \bar{k}, c_1, c_2]$;
      Compute $\delta(\bar{k}) \leftarrow \max\{V_d(\mathcal{I}, a, \bar{k}) - U, 1\}$; # *at least reduce $\bar{k}$, so $\delta(\bar{k}) \geq 1$*
      $\mathbb{K} \leftarrow \{k \mid k \geq$ `searchfirst`$(\bar{k} + \delta(\bar{k}), \mathbb{K}), k \in \mathbb{K}\}$; # *jump the reduced and repeated elements*
return: Optimal loss U and decision tree T.

When $d = 1$, the `BR` method reduces to a special case, which is outlined in Algorithm 5. In this case, the reduction strategy in Algorithm 2 is also applied to `CART` to solve `ABR`$(\mathcal{I}, 1)$. The reduced `CART` is named as `R-CART`, which is similar to a special case of Algorithm 2 to depth-1 trees.

Since MHABR involves multiple uses of CART, even a modest improvement in CART would significantly boost efficiency and have a cumulative effect on the overall algorithm. The reduction strategy provides a more efficient gain for $d = 2$ compared to $d = 1$, as the variability in subtree loss increases with depth. In the absence of reduction, Algorithm 2 converges to the global optimum of Equation (3) within at most $\sum_{a=1}^{p} n_a$ evaluations of CART. Unlike our BR method, CART follows a forward search approach rather than using a bisection branching strategy. Notably, the R-CART algorithm can be viewed as a special case of the BR method proposed in Algorithm 2, corresponding to a depth-1 tree without recursion for lower depths.

8.3 An Illustrative XOR Case for Greedy Failure

In this section, we use XOR-style datasets as an illustrative example to explain why a lookahead-based method can outperform greedy CART. The XOR problem is a canonical topological trap for greedy splitting: a depth-1 criterion may fail to identify informative features, whereas a shallow lookahead can recover the correct interaction. Our goal here is to build intuition for the behavior of MHABR.

More broadly, ABR can improve upon CART in settings where a split that appears suboptimal under a purely greedy criterion leads to a better downstream subtree configuration under lookahead. The following XOR example provides a simple instance of this phenomenon.

XOR-Distractor Dataset. We consider the following stylized data distribution.

Definition 8.1. Let the dataset be given by $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \{0,1\}^p$ and $y_i \in \mathcal{Y} = \{0,1\}$. The target y_i is determined exclusively by the first two features via the XOR function: $y_i = x_i^1 \oplus x_i^2$, where x_i^1 and x_i^2 are drawn from independent uniform Bernoulli distributions ($P = 0.5$). The remaining features $x_i^3, \ldots, x_i^p$ are pure noise, drawn independently from $P(x_i^h = 1) = 0.5$ for $h \geq 3$.

In this setting, a single greedy split on either informative feature does not reduce the immediate misclassification loss, because each informative feature alone leaves the label distribution balanced within the resulting child nodes. In contrast, a depth-2 lookahead can recover the XOR interaction by selecting splits that jointly isolate the homogeneous quadrants.

Theorem 8.2 (Failure of Greedy Splits)**.** *Given an XOR-Distractor dataset defined in Definition 8.1, let $V_0(\mathcal{I})$ denote the misclassification loss of a root node containing n samples. Under a finite sample size n, empirical sampling variance introduces a perturbation $\epsilon > 0$ to the class distribution of random features. A greedy top-down decision tree algorithm will deterministically select a noise feature $a = h$ ($h \geq 3$) over a true feature $a = 1$ or $a = 2$ at the root node.*

Proof. Let $\mathcal{I}$ be the sample index set at the root node, where $|\mathcal{I}| = n$. Due to the uniform distribution of the true features, the marginal probability of the target is perfectly balanced: $P(y = 1) = 0.5$. Therefore, the initial misclassification loss is $V_0(\mathcal{I}) = 0.5n$.

Consider a candidate split on the true feature $a = 1$. The data is partitioned into $\mathcal{I}_L^1 = \{i \in \mathcal{I} \mid x_i^1 = 0\}$ and $\mathcal{I}_R^1 = \{i \in \mathcal{I} \mid x_i^1 = 1\}$. In $\mathcal{I}_L^1$, the target becomes $y_i = 0 \oplus x_i^2 = x_i^2$. Since x_i^2 is uniformly distributed, $P(y = 1 \mid x_i^1 = 0) = 0.5$. Similarly, in $\mathcal{I}_R^1$, $y_i = 1 \oplus x_i^2 = 1 - x_i^2$, yielding $P(y = 1 \mid x_i^1 = 1) = 0.5$. The loss reduction ($\Delta V_0$) for the true feature is:

$$\begin{aligned}\Delta V_0(a = 1) &= V_0(\mathcal{I}) - \big(V_0(\mathcal{I}_L^1) + V_0(\mathcal{I}_R^1)\big) \\ &= 0.5n - \big(0.5|\mathcal{I}_L^1| + 0.5|\mathcal{I}_R^1|\big) \\ &= 0.5n - 0.5n = 0\end{aligned}$$

Now consider a candidate split on a noise feature $a = h$ ($h \geq 3$). Due to finite sample variance, the empirical distribution of y within the resulting partitions $\mathcal{I}_L^h$ and $\mathcal{I}_R^h$ will deviate from exactly 0.5 by some small noise factor $\epsilon > 0$. Assume the majority class proportion in $\mathcal{I}_L^h$ becomes $0.5 + \epsilon$. The misclassification loss for $\mathcal{I}_L^h$ is determined by the minority proportion: $V_0(\mathcal{I}_L^h) = (0.5 - \epsilon)|\mathcal{I}_L^h|$.

Assuming a symmetric perturbation in $\mathcal{I}_R^h$, the expected loss reduction for the noise feature evaluates strictly greater than zero:

$$\begin{aligned}\Delta V_0(a=h) &= 0.5n - \big((0.5-\epsilon)|\mathcal{I}_L^h| + (0.5-\epsilon)|\mathcal{I}_R^h|\big) \\ &= 0.5n - (0.5-\epsilon)n \\ &= \epsilon n > 0\end{aligned}$$

Since $\Delta V_0(a=h) > \Delta V_0(a=1)$, the greedy algorithm maximizes local error reduction by splitting on the distractor feature $a=h$, trapping the tree in a suboptimal local minimum. □

Illustrative role of lookahead For XOR-style data, a depth-2 tree can represent the target rule exactly, whereas a purely greedy depth-1 criterion may not discover the required structure. This makes XOR an example for visualizing the difference between greedy splitting and lookahead. Figure 9 illustrates this contrast.

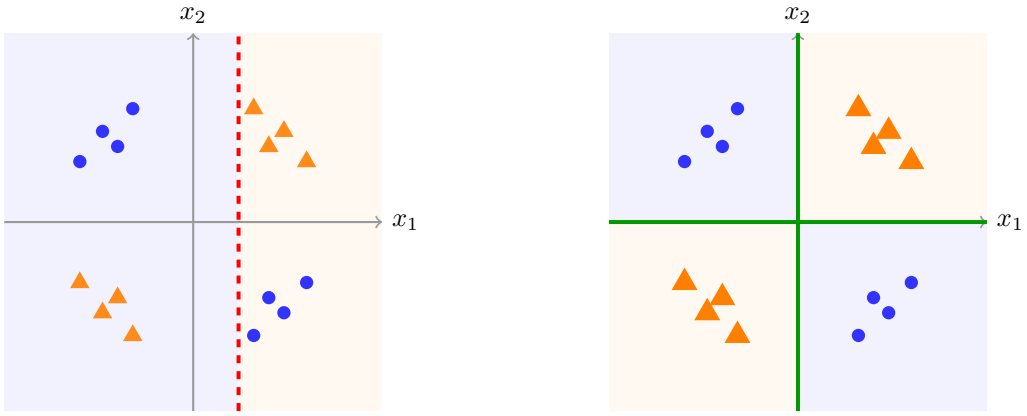


Figure 9: Illustration of a canonical XOR failure mode. A depth-1 greedy split may fail to separate the classes, whereas a depth-2 lookahead can recover the correct interaction structure.

This example highlights a basic limitation of greedy splitting: the usefulness of a feature may only become apparent after subsequent splits are taken into account. A lookahead-based method such as `ABR` is designed to evaluate such downstream effects more explicitly.

Theorem 8.3. *Let $\mathcal{D}$ be an XOR-distributed dataset augmented with continuous noise features. On such a topology, the distance-based reduction mechanism of `MHABR` is mathematically guaranteed to never remove the global optimum from the candidate set. Consequently, `MHABR` safely evaluates the exact ground-truth interaction, yielding a strictly superior solution compared to `CART`.*

Proof. Let W_d denote the misclassification loss of the initial tree generated by `CART`, establishing the initial global upper bound $U = W_d$ for the `MHABR` algorithm. Let k^* denote the candidate split configuration representing the true optimal topological interaction (splitting on the ground-truth features x_1 and x_2). We aim to prove that k^* is never removed by the reduction strategy defined in Lemma 3.3.

By the definition of an XOR distribution, the target variable y is perfectly balanced across any single marginal axis. Thus, the immediate impurity reduction for any depth-1 split is strictly zero. Relying on a greedy depth-1 lookahead, `CART` fails to observe the depth-2 interaction and splits arbitrarily on noise features, generating a highly impure tree. Consequently, the initial global upper bound is strictly positive: $U = W_d > 0$.

Conversely, the ground-truth interaction k^* correctly partitions the sample space into four pure homogeneous quadrants. Because the class variance within each resulting leaf is zero, the exact evaluation of this lookahead topology yields the absolute minimum misclassification loss: $\hat{V}_d(k^*) = 0$.

During the search, suppose the algorithm evaluates a midpoint $\bar{k}$. According to Lemma 3.2, the loss function satisfies a Lipschitz-like continuity over the sorted index space, meaning the change in loss is bounded by the index distance:

$$\hat{V}_d(\bar{k}) - \hat{V}_d(k^*) \leq |\zeta(\bar{k}) - \zeta(k^*)|$$

Substituting $\hat{V}_d(k^*) = 0$, we establish a strict structural bound on the evaluated loss:

$$\hat{V}_d(\bar{k}) \leq |\zeta(\bar{k}) - \zeta(k^*)| \tag{28}$$

According to the reduction strategy (Lemma 3.3), a candidate split is removed from the search space if and only if its distance to $\bar{k}$ is less than or equal to the reduction margin $\delta(\bar{k}) := \hat{V}_d(\bar{k}) - U$.

Given that $U > 0$ due to the heuristic failure of `CART`, it follows trivially that:

$$\delta(\bar{k}) = \hat{V}_d(\bar{k}) - U < \hat{V}_d(\bar{k})$$

Combining this with Inequality Equation 28, we obtain:

$$\delta(\bar{k}) < |\zeta(\bar{k}) - \zeta(k^*)|$$

Because the reduction margin $\delta(\bar{k})$ is strictly less than the distance to the optimal split, the required condition for pruning ($|\zeta(\bar{k}) - \zeta(k^*)| \leq \delta(\bar{k})$) is never satisfied. Therefore, the reduction mechanism maintains the optimal solution within the active candidate set. `MHABR` is formally guaranteed to evaluate k^*, update the global upper bound to 0, and return a solution strictly superior to `CART`. □

9 Details of Numerical Experiments

9.1 Information of small-scale datasets

The details of the 51 small datasets are presented in Table 8.

9.2 Configuration of additional techniques for large-scale datasets

The results of these datasets with a sample size greater than $10,000$ are given as follows. For the 5 medium datasets, we do not introduce the parameters ε and θ; `MHABR` successfully completes all the training tasks. For the three large-scale datasets, we configure the algorithms to terminate within the time limit. The configurations are SUSY: $\varepsilon = 0.01, \theta = 0.25$, HIGGS: $\varepsilon = 0.01, \theta = 0.5$, WESAD: $\varepsilon = 0.00025, \theta = 0.25$ respectively.

9.3 Implementation details

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.

MHABR: Our algorithm is implemented in `Julia`. For the default version, we adopt the reduction strategy with parameters $\varepsilon = 0$ and $\theta = 1$.

CART (Sadeghi et al., 2022): We use the implementation from the `Julia` package `DecisionTree`, selecting entropy loss as it provides the best results among the three options; its performance may occasionally surpass `MHABR` and `DPDT` on several datasets.

DPDT (Kohler et al., 2025): This algorithm is written in Python and calls `CART` by `Python` package `sklearn`, using the **Gini loss** (default). Therefore, in our results, it may be surpassed by `CART`.

Table 8: The information of 51 small-scale datasets.

Dataset Name	n	p	Class	Dataset Name	n	p	Class
Soybean-small	47	35	4	Body	507	5	2
Echocardiogram	61	11	2	Climate-model-crashes	540	20	2
Hepatitis	80	19	2	Monks-problems-3	554	6	2
Fertility	100	9	2	Monks-problems-1	556	6	2
Acute-inflammations-1	120	6	2	Breast-cancer-diagnosti	569	30	2
Acute-inflammations-2	120	6	2	Monks-problems-2	600	6	2
Hayes–roth	132	5	3	Balance-scale	625	4	3
Iris	150	4	3	Credit-approval	653	15	2
Teaching-assistant-evaluation	151	5	3	Breast-cancer	683	9	2
Wine	178	13	3	Blood-transfusion	748	4	2
Breast-cancer-prognostic	194	31	2	Mammographic-mass	830	5	2
Parkinsons	195	23	2	Tic-tac-toe-endgame	958	9	2
Connectionist-bench-sonar	208	60	2	Connectionist-bench	990	13	11
Image-segmentation	210	19	7	Statlog-project-German-credit	1,000	20	2
Seeds	210	7	3	Concrete	1,030	8	3
Glass	214	9	6	Banknote-authentication	1,372	4	2
Thyroid-disease-new-thyroid	215	5	3	Contraceptive-method-choice	1,473	9	3
Congressional-voting-records	232	16	2	Car-evaluation	1,728	6	4
Spect-heart	267	22	2	Ozone-level-detection-eight	1,847	72	2
Spectf-heart	267	44	2	Ozone-level-detection-one	1,848	72	2
Cylinder-bands	277	39	2	Seismic-bumps	2,584	18	2
Heart-disease-Cleveland	282	13	5	Chess-king-rook-versus-king-pawn	3,196	36	2
Haberman-survival	306	3	2	Thyroidann	3,772	21	3
Ionosphere	351	34	2	Wall-following-robot-2	5,456	2	4
Dermatology	358	34	6	Thyroid-disease-ann-thyroid	7,200	21	3
Thoracic-surgery	470	16	2				

LS-OCT (Dunn, 2018): Since the original code is not available, we implement both methods in `Julia` and call `Gurobi` to solve MIP models.

DL8.5 (Aglin et al., 2020): This algorithm is written in `C++` and is run as an extension of `Python`. The current version also integrates the methods of `MurTree` (Demirović et al., 2022). Besides, we utilize the `GUESS` (McTavish et al., 2022) for binarization, which has the best performance.

Quant-BnB (Mazumder et al., 2022): The authors provide an open-source implementation of this algorithm, which is written in `Julia`.

TAO (Carreira-Perpinán & Tavallali, 2018): Because the source code for this method is not publicly available, we directly compare our approach with the results reported in the original paper under the same experimental configuration.

Additionally, we experimented with **STreeD** (van der Linden et al., 2023), implemented in `C++` with a `Python` interface. However, due to changes in computational hardware, a fair comparison could not be ensured; therefore, we omit its results.