Shattering Inequalities for Learning Optimal Decision Trees

Justin Boutilier[1], Carla Michini[1], Zachary Zhou[1*]

[1*]Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI, USA.

*Corresponding author(s). E-mail(s): zzhou246@wisc.edu;
Contributing authors: jboutilier@wisc.edu; michini@wisc.edu;

Abstract

Recently, mixed-integer programming (MIP) techniques have been applied to learn optimal decision trees. Empirical research has shown that optimal trees typically have better out-of-sample performance than heuristic approaches such as CART. However, the underlying MIP formulations often suffer from weak linear programming (LP) relaxations. Many existing MIP approaches employ big-M constraints to ensure observations are routed throughout the tree in a feasible manner. This paper introduces new MIP formulations for learning optimal decision trees with multivariate branching rules and no assumptions on the feature types. We first propose a strong baseline MIP formulation that still uses big-M constraints, but yields a stronger LP relaxation than its counterparts in the literature. We then introduce a problem-specific class of valid inequalities called shattering inequalities. Each inequality encodes an inclusion-minimal set of points that cannot be shattered by a multivariate split, and in the context of a MIP formulation, the inequalities are sparse, involving at most the number of features plus two variables. We propose a separation procedure that attempts to find a violated inequality given a (possibly fractional) solution to the LP relaxation; in the case where the solution is integer, the separation is exact. Numerical experiments show that our MIP approach outperforms two other MIP formulations in terms of solution time and relative gap, and is able to improve solution time while remaining competitive with regards to out-of-sample accuracy in comparison to a wider range of approaches from the literature.

Keywords: Decision Trees, Mixed-Integer Programming, Machine Learning

1 Introduction

Decision trees are among the most popular techniques for interpretable machine learning [1]. In addition to their use as a standalone method, decision trees form the foundation for several more sophisticated machine learning algorithms such as random forest [2, 3]. Although there are many ways to express a decision tree, the majority of the literature, including this paper, focuses on binary trees. In a binary decision tree, each internal node, referred to as a *branch node*, has exactly two children and observations are routed to the left or right child node according to a *branching rule*. Terminal nodes in the tree are referred to as *leaf nodes* and each leaf node is assigned a class k such that any observation routed to that leaf node is classified as belonging to class k. Almost all algorithms (heuristic and exact) that generate decision trees focus on *univariate* branching rules, which check if the value of a single feature exceeds a prescribed threshold. In this work, we instead focus on *multivariate* branching rules, which are separating hyperplanes checking several features at a time. Multivariate branching rules are less easily interpretable, however they provide more flexibility than univariate branching rules, which can only resort to axis-aligned hyperplanes. As a consequence, multivariate branching rules can yield much more compact decision trees. In other words, even if the tests performed at the branching nodes are more complex, the total number of tests needed to achieve a target accuracy can be dramatically smaller; see the toy example in Figure 1.

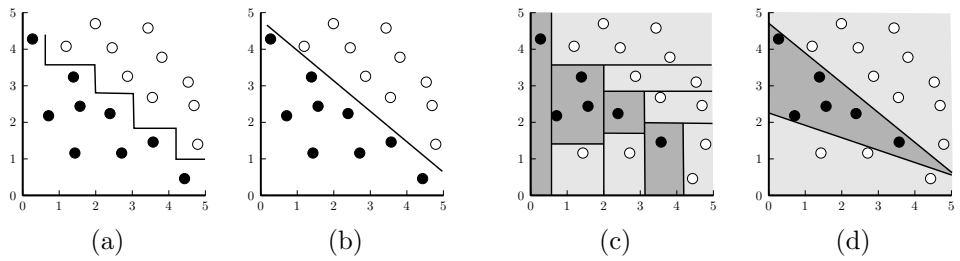


Fig. 1 (a) Eight univariate branching rules are required to correctly separate the black and white observations; (b) only one multivariate branching rule suffices. For a slightly different toy dataset: (c) and (d) show how the feature space is partitioned when using univariate and multivariate branching rules, respectively.

Related work. The problem of learning optimal decision trees is NP-hard [4]. As a result, there exist several famous top-down induction algorithms for learning decision trees such as CART [1] and ID3 [5]. These heuristic methods do not provide any guarantee on the quality of the decision trees computed. More recently, a number of exact approaches have been proposed that typically aim at minimizing the training error and possibly some measure of the tree complexity.

One stream of work uses mixed-integer programming (MIP) to compute optimal decision trees. Motivated by algorithmic advances in integer optimization, Bertsimas and Dunn [6] first formulated the problem of learning an optimal decision tree as a MIP. Their work spurred a series of subsequent papers that propose a variety of

MIP tools to model and solve the problem of learning optimal decision trees [7–16]. One main advantage of MIP approaches is their flexibility: the problem objective can be easily modified to enhance feature selection and/or tree size, and the feasible set can be modified by adding additional constraints of practical interest [7, 9, 11, 13]. Moreover, MIP formulations can easily handle multivariate branching rules [6, 16]. Typically, MIP formulations contain two key components: 1) a framework to model the routing of observations through the decision tree to leaf nodes, and 2) a framework that properly constructs the tree by devising branching rules at each branch node. Most MIP approaches employ big-M constraints to unify these two components in a single optimization problem [6, 15, 16], but this modeling technique suffers from the fact that big-M constraints notoriously lead to poor LP relaxations [17]. One notable exception is the work of Aghaei et al. [8], who consider datasets with binary features and formulate the problem of routing observations through a fixed decision tree with univariate branching rules as a max-flow problem. Thanks to these two key assumptions – having binary features and restricting to univariate branching rules – they can avoid using big-M constraints. Moreover, their formulation is amenable to a Benders decomposition, where the master problem is tasked with constructing the decision tree, and the routing of observations to leaf nodes is accomplished by solving a subproblem for each observation that adds optimality cuts to the master. Unfortunately, their flow-based approach does not generalize if we relax either of the two key assumptions.

Another stream of work uses Boolean satisfiability (SAT) [18–21], constraint programming (CP) [22, 23], and dynamic programming [24–29] to compute optimal decision trees. These methods address the scalability issues of general MIP methods by using a different range of techniques to explore the search space, such as sub-sampling, branch and bound search, and caching. Some of these algorithms are tailored to the specific structure of decision trees, which is used to speed-up computation. However, 1) all of them assume binary features or use binarization techniques to transform numerical features into binary ones in a preprocessing step, which can dramatically increase the size of the input (causing memory problems) and is not guaranteed to preserve optimality [30]; 2) most of them are designed and/or implemented to work only for binary classification; and 3) all of them are only suited to construct decision trees with univariate branching rules.

Our contribution. We first propose a new MIP formulation for learning optimal decision trees with multivariate branching rules and no assumptions on the feature types. Our formulation employs only binary variables to (i) express how each observation is routed in the decision tree and (ii) express the objective function as a weighed sum of the training accuracy and the size of the tree. Moreover, we exploit the structure of decision trees and use a geometric interpretation of the optimal decision tree problem to devise a specialized class of valid inequalities, called *shattering* inequalities, which intuitively detect problematic sub-samples of the dataset that cannot be linearly separated. We leverage these inequalities within a Benders-like decomposition [31] to decompose our formulation into a master problem that determines how to route each observation to a leaf node, and a collection of linear programming (LP) feasibility subproblems that certify whether, for each branch node of the decision tree, it is

possible to construct a multivariate branching rule that realizes the given routing of observations. If it is not possible to realize the routing, then we add one of our valid inequalities to the master problem as a feasibility cut. Each of our inequalities encodes a minimal set of points that cannot be shattered by a multivariate branching rule and in the context of a MIP formulation, the inequalities are sparse, with at most the number of features plus two variables. Our approach does not require big-M constraints, but generates sparse cuts that capture the combinatorial structure of the problem to strengthen the LP relaxation and decrease training time. Although we use these cuts in a decomposition algorithm for our formulation, they can be directly applied as valid inequalities to other MIP formulations that may not be suited to decomposition (e.g., OCT-H [6] and SVM1-ODT [16]). We demonstrate through numerical experiments that our MIP approach outperforms (in terms of training accuracy, testing accuracy, solution time, and relative gap) two other popular MIP formulations, and is able to improve both in and out-of-sample performance, while remaining competitive in terms of solution time to a wide range of popular approaches from the literature.

2 The Optimal Decision Tree Problem

In this section, we formally introduce the problem setting, our notation, and our formulation.

2.1 The Optimal Decision Tree Problem

We first define our data, which includes a training set of N observations, each of which has p numerical features and belongs to one of K classes. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. Without loss of generality, we normalize the training set so that all features are scaled to $[0, 1]$. Thus, each observation $i \in [N]$ is a vector $(\mathbf{x}^i, y^i) \in [0, 1]^p \times [K]$.

As noted in Section 1, we focus on learning optimal binary decision trees with multivariate branching rules, which we refer to as *multivariate splits*. Each node in the tree is either a branch node or a leaf node. Note that the first node in the tree is colloquially referred to as the root node (even though it is a branch node). The model is built upon a binary tree where every branch node has exactly two children and the leaves are all on the same level. The maximum depth of the decision tree $D \in \mathbb{N}$ is defined as the length of the path from the root to any leaf. We denote the set of branch nodes as $\mathcal{B} = \{1, \dots, 2^D - 1\}$. Each branch node t corresponds to a multivariate split defined by learned parameters $\mathbf{a}_t \in \mathbb{R}^p$ and $b_t \in \mathbb{R}$. The multivariate split is applied as follows: for each observation $\mathbf{x} \in [0, 1]^p$, if $\mathbf{a}_t^\top \mathbf{x} \leq b_t$, then $\mathbf{x}$ is sent to the left child of t, denoted by $2t$; otherwise it is sent to the right child, denoted by $2t + 1$. The key difference between multivariate and univariate splits is that a univariate split allows only one component of $\mathbf{a}_t$ to be non-zero. We denote the set of leaf nodes by $\mathcal{L} = \{2^D, \dots, 2^{D+1} - 1\}$. Each leaf node t is a terminal node (i.e., it has no children) and is assigned a class $k \in [K]$. All observations routed to leaf t are classified as belonging to class k.

The max depth $D \in \mathbb{N}$ is often used as a hyperparameter to control the complexity and size of the tree. To deter the model from constructing a full binary tree of depth

D, we use a hyperparameter $\beta \in \mathbb{N}$ to limit the number of non-trivial splits used (more on this in Section 2.2).

2.2 Problem Formulation

We now present a formulation for learning an optimal decision tree – the training problem – that includes a set of complicating constraints. For each branch node $t \in \mathcal{B}$, we can either define a branching rule establishing whether an incoming observation should be sent to the left or to the right child of t, in which case we say that *node t applies a (non-trivial) split*, or we can direct all of the incoming observations to the left child of t. Correspondingly, we introduce a binary variable d_t that is equal to 1 if t applies a split, and to 0 otherwise. The decision variables $\mathbf{d}$ thus define the tree topology. For each $t \in \mathcal{B}$ we have $p+1$ variables $(\mathbf{a}_t, b_t)$ defining the multivariate split associated with the branch node. If t does *not* apply a split, it is feasible to set these variables to $(\mathbf{0}, 0)$.

For each observation $i \in [N]$ and for each node $t \in \mathcal{B} \cup \mathcal{L}$ of the decision tree, we introduce a binary variable w_{it} that is equal to 1 if observation i is sent to node t, and to 0 otherwise. The decision variables $\mathbf{w}$ thus define how to route the observations from the root node to the leaf nodes.

For each class $k \in [K]$ and leaf node $t \in \mathcal{L}$, we introduce a binary decision variable c_{tk} that is equal to 1 if t is assigned class label k, and to 0 otherwise. Finally, for each observation $i \in [N]$ and leaf node $t \in \mathcal{L}$, we introduce a binary-valued decision variable z_{it} that is equal to 1 if i is sent to leaf t and is correctly classified as y^i, and to 0 otherwise. We will later see that the integrality constraints on $\mathbf{z}$ can be relaxed. Our formulation for the training problem is

$$
\begin{aligned}
\underset{\mathbf{c},\mathbf{d},\mathbf{w},\mathbf{z},\mathbf{a},\mathbf{b}}{\text{maximize}} \quad & \sum_{i=1}^{N} \sum_{t \in \mathcal{L}} z_{it} && && \text{(1a)}\\
\text{subject to} \quad & \sum_{t \in \mathcal{L}} w_{it} = 1 && \forall i \in [N], && \text{(1b)}\\
& w_{it} = w_{i,2t} + w_{i,2t+1} && \forall i \in [N],\ t \in \mathcal{B}, && \text{(1c)}\\
& \sum_{k=1}^{K} c_{tk} = 1 && \forall t \in \mathcal{L}, && \text{(1d)}\\
& z_{it} \leq w_{it} && \forall i \in [N],\ t \in \mathcal{L}, && \text{(1e)}\\
& z_{it} \leq c_{t,y^i} && \forall i \in [N],\ t \in \mathcal{L}, && \text{(1f)}\\
& w_{i,2t+1} \leq d_t && \forall i \in [N],\ t \in \mathcal{B}, && \text{(1g)}\\
& \sum_{t \in \mathcal{B}} d_t \leq \beta, && && \text{(1h)}\\
& c_{tk} \in \{0,1\} && \forall k \in [K],\ t \in \mathcal{L}, && \text{(1i)}\\
& d_t \in \{0,1\} && \forall t \in \mathcal{B}, && \text{(1j)}\\
& w_{it} \in \{0,1\} && \forall i \in [N],\ t \in \mathcal{B} \cup \mathcal{L}, && \text{(1k)}
\end{aligned}
$$

$$z_{it} \in \mathbb{R} \qquad \forall i \in [N],\ t \in \mathcal{L}, \tag{1l}$$
$$(\mathbf{a}_t, b_t) \in \mathcal{H}_t(\mathbf{w}) \qquad \forall t \in \mathcal{B}, \tag{1m}$$

where, for each branch node $t \in \mathcal{B}$ and integral $\mathbf{w}$ satisfying (1b) and (1c), the set $\mathcal{H}_t(\mathbf{w})$ is defined as

$$\mathcal{H}_t(\mathbf{w}) = \big\{(\mathbf{a}_t, b_t) \in \mathbb{R}^{p+1} : \mathbf{a}_t^\top \mathbf{x}^i + 1 \le b_t \qquad \forall\, i \in [N] : w_{i,2t} = 1, \tag{2}$$
$$\mathbf{a}_t^\top \mathbf{x}^i - 1 \ge b_t \qquad \forall\, i \in [N] : w_{i,2t+1} = 1\big\}. \tag{3}$$

Note that, for a fixed $\mathbf{w}$, the set $\mathcal{H}_t(\mathbf{w})$ is a (possibly empty) polyhedron in $\mathbb{R}^{p+1}$.

The objective function (1a) aims to maximize the total number of observations that are correctly classified. Our code implementation allows the user to specify a regularization hyperparameter $\alpha \ge 0$ such that the objective is changed to mirror CART's cost-complexity measure [1]: minimize $\frac{1}{N}\sum_{i=1}^{N}\big(1 - \sum_{t\in\mathcal{L}} z_{it}\big) + \alpha \sum_{t\in\mathcal{B}} d_t$. Constraints (1b) ensure that each observation is mapped to exactly one leaf, while constraints (1c) guarantee that each observation routed to a branch node t is sent to either the left or the right child of t. For a branch node t that does not apply a split, constraints (1g) automatically send any incoming observations to the left child of t. Constraints (1d) assign each leaf node a class in $[K]$. Constraints (1e) and (1f) enforce the condition that if $z_{it} = 1$, then $w_{it} = 1$ and $c_{t,y^i} = 1$ (i.e., observation i is sent to leaf t and is correctly classified as y^i). Constraint (1h) limits the number of non-trivial splits used to be at most β, where $\beta \in \{1, \ldots, 2^D - 1\}$. The main motivation for introducing hyperparameter β is that, while the search space for α is infinitely large, for β we only need to search over finitely many choices [6]. We remark that integrality constraints are not required for the $\mathbf{z}$ and $\mathbf{c}$ variables, since they are implied by the integrality of $\mathbf{w}$ and by the fact that at an optimal solution constraints (1e) and (1f) hold with equality, see [32] for a formal proof. Complicating constraints (1m) are the only ones involving variables $(\mathbf{a}_t, b_t),\ t \in \mathcal{B}$, which ensure that the routing defined by $\mathbf{w}$ can be *realized* by multivariate splits. This is possible if and only if for each branch node t we have $\mathcal{H}_t(\mathbf{w}) \neq \varnothing$.

We highlight some technical differences between our formulation and other MIP models in the literature. First, we define the set of feasible routings $\mathbf{w}$ as the routings that satisfy constraints (1m). As it will be clear in the following, we have a range of options to express $\mathcal{H}_t(w)$, i.e., by resorting to big-M constraints, by dynamically adding violated shattering inequalities, or with a mix of these two approaches. Second, we define $\mathbf{w}$ over all nodes of the decision tree, while previous formulation defines $\mathbf{w}$ over only the leaf nodes [6, 13, 16]. Our primary motivation for defining additional (roughly double) $\mathbf{w}$ variables is that we can exploit these additional variables to create stronger valid inequalities for characterizing the set of feasible routings, see Section 4. A secondary reason for the introduction of $\mathbf{w}$ variables over the branch nodes is that these extra variables give us the option to formulate a model using fewer and stronger big-M constraints, see Section 3.

Finally, we maximize training accuracy and we control the tree size by imposing constraint (1h). Unlike the univariate setting where CART's cost-complexity measure can be directly used as a template, the multivariate setting has no universally accepted

objective. For example, OCT-H [6] penalizes the total number of *features* used over all splits in the tree and SVM1-ODT [16] penalizes the ℓ_1 norm of $\mathbf{a}_t$ over all splits in the tree.

3 Baseline MIP formulation

For a fixed branch node $t \in \mathcal{B}$, the complicating constraint (1m) holds if and only if for every observation $i \in [N]$, the logical constraints

$$w_{i,2t} = 1 \implies \mathbf{a}_t^\top \mathbf{x}^i \leq b_t, \tag{4a}$$
$$w_{i,2t+1} = 1 \implies \mathbf{a}_t^\top \mathbf{x}^i - \epsilon \geq b_t \tag{4b}$$

are satisfied for some sufficiently small $\epsilon > 0$[1]. Typically, logical implications are modeled using big-M constraints; indeed, this is the approach used by many optimal tree formulations for imposing (1m) [6, 14–16]. To ensure the quantity $\mathbf{a}_t^\top x^i - b_t$ is bounded so that a big-M value can be defined, we enforce the conditions $\|\mathbf{a}_t\|_1 \leq 1,\ b_t \in [-1, 1]$. Thus, we have the constraints

$$\sum_{j=1}^{p} \hat{a}_{tj} \leq 1, \tag{5a}$$
$$\hat{a}_{tj} \geq a_{tj} \qquad j \in [p], \tag{5b}$$
$$\hat{a}_{tj} \geq -a_{tj} \qquad j \in [p], \tag{5c}$$
$$b_t \in [-1, 1], \tag{5d}$$

where the auxiliary variables $\hat{a}_{tj}$ are used to model $|a_{tj}|$ for each $j \in [p]$. We obtain from (4) the big-M constraints

$$\mathbf{a}_t^\top \mathbf{x}^i \leq b_t + M_i(1 - w_{i,2t}) \qquad i \in [N], \tag{6a}$$
$$\mathbf{a}_t^\top \mathbf{x}^i - \epsilon \geq b_t - (M_i + \epsilon)(1 - w_{i,2t+1}) \qquad i \in [N], \tag{6b}$$

with big-M values

$$M_i = \max_{j \in [p]} \{x_j^i\} + 1 \quad \forall i \in [N]. \tag{7}$$

We define our *baseline* formulation to be (1) where the complicating constraints (1m) are implemented using big-M constraints (6) for all $t \in \mathcal{B}$.

We point out that some MIP formulations found in the literature do not employ big-M constraints. This is typically the case of univariate decision tree models relying on the assumption that the feature vectors $\mathbf{x}$ are binary [8, 9, 11]. However, in the absence of either of these two assumptions—univariate splits or binary features—it becomes more complicated to circumvent the use of big-M constraints.

We provide a comparison with how the OCT-H formulation of Bertsimas and Dunn [6] models (1m), and argue our formulation yields a stronger LP relaxation. In the

[1] In practice, ϵ is fixed in advance; a reasonable choice is $\epsilon = 0.005$.

OCT-H formulation, the variables w_{it} are defined only for the leaf nodes $t \in \mathcal{L}$. The authors define the set $\mathcal{A}_L(t)$ (resp. $\mathcal{A}_R(t)$) of ancestors of leaf node $t \in \mathcal{L}$ whose left (resp. right) child is in the path from the root node to node t. As a consequence, the logical constraints implemented in OCT-H are

$$w_{it} = 1 \implies \mathbf{a}_m^\top \mathbf{x}^i \leq b_m \qquad \forall i \in [N],\ t \in \mathcal{L},\ m \in \mathcal{A}_L(t), \tag{8a}$$

$$w_{it} = 1 \implies \mathbf{a}_m^\top \mathbf{x}^i - \epsilon \geq b_m \qquad \forall i \in [N],\ t \in \mathcal{L},\ m \in \mathcal{A}_R(t). \tag{8b}$$

These logical constraints are then linearized to obtain the big-M constraints[2]

$$\mathbf{a}_m^\top \mathbf{x}^i \leq b_m + M_i'(1 - w_{it}) \qquad \forall i \in [N],\ t \in \mathcal{L},\ m \in \mathcal{A}_L(t), \tag{9a}$$

$$\mathbf{a}_m^\top \mathbf{x}^i - \epsilon \geq b_m - (M_i' + \epsilon)(1 - w_{it}) \qquad \forall i \in [N],\ t \in \mathcal{L},\ m \in \mathcal{A}_R(t), \tag{9b}$$

with big-M values

$$M_i' = 2 \quad \forall i \in [N]. \tag{10}$$

The big-M values (10) exploit only the assumption that features are normalized to the $[0, 1]$ interval, while the big-M values (7) are specific of each observation $i \in [N]$. Since for most observations $i \in [N]$, $\max_{j \in [p]}\{x_j^i\}$ may be much smaller than 1, the big-M values in the OCT-H formulation are not as tight as the big-M values (7) utilized in our baseline formulation.

Second, and more importantly, the big-M constraints of OCT-H result in weaker LP relaxations. We denote by P_{base} and $P_{\text{OCT-H}}$ the linear relaxations of our baseline formulation and of OCT-H, respectively. We consider the projections of P_{base} and $P_{\text{OCT-H}}$ on variables $\{w_{it} : i \in [N], t \in \mathcal{L}\}$ and $\{(a_t, b_t) : t \in \mathcal{B}\}$, which we denote by Q_{base} and $Q_{\text{OCT-H}}$, respectively. We consider the case where $\beta = 2^D - 1$ and there is no restriction on the minimum number of observations in leaf nodes, which is an additional parameter of OCT-H.

Proposition 1. *Q_{base} is strictly contained in $Q_{OCT\text{-}H}$.*

Proof. The big-M constraints (9) in OCT-H can be rewritten as

$$\mathbf{a}_t^\top \mathbf{x}^i \leq b_t + M_i'(1 - w_{i,\hat{t}}) \qquad \forall i \in [N],\ t \in \mathcal{B},\ \hat{t} \in \mathcal{L}(2t), \tag{11a}$$

$$\mathbf{a}_t^\top \mathbf{x}^i - \epsilon \geq b_t - (M_i' + \epsilon)(1 - w_{i,\hat{t}}) \qquad \forall i \in [N],\ t \in \mathcal{B},\ \hat{t} \in \mathcal{L}(2t+1), \tag{11b}$$

where $\mathcal{L}(t)$ denotes the set of leaf nodes in the subtree whose root is $t \in \mathcal{B}$. Based on the discussion in [6], it is evident that $Q_{\text{OCT-H}}$ is determined by $w_{it} \geq 0$ for all $i \in [N]$, $t \in \mathcal{L}$, (1b), (5) and (11). This is because, for each assignment of $\{w_{it} : i \in [N], t \in \mathcal{L}\}$ and $\{(a_t, b_t) : t \in \mathcal{B}\}$ satisfying these constraints we can derive values of the other problem variables that are feasible for $P_{\text{OCT-H}}$.

[2] In [6], ϵ appears in first set of constraints (9a), not in the second set of constraints (9b) as we have written. For the sake of comparison, we put ϵ in (9b).

In our formulation, $w_{it} = \sum_{t' \in \mathcal{L}(t)} w_{i,t'}$ for all $i \in [N],\ t \in \mathcal{B}$, so by writing our big-M constraints (6) for all $t \in \mathcal{B}$, we obtain

$$\mathbf{a}_t^\top \mathbf{x}^i \leq b_t + M_i \left(1 - \sum_{t' \in \mathcal{L}(2t)} w_{i,t'}\right) \qquad \forall i \in [N],\ t \in \mathcal{B}, \tag{12a}$$

$$\mathbf{a}_t^\top \mathbf{x}^i - \epsilon \geq b_t - (M_i + \epsilon) \left(1 - \sum_{t' \in \mathcal{L}(2t+1)} w_{i,t'}\right) \qquad \forall i \in [N],\ t \in \mathcal{B}. \tag{12b}$$

Q_{base} is determined by $w_{it} \geq 0$ for all $i \in [N]$, $t \in \mathcal{L}$, (1b), (5) and (12), since for each assignment of $\{w_{it} : i \in [N], t \in \mathcal{L}\}$ and $\{(a_t, b_t) : t \in \mathcal{B}\}$ satisfying these constraints we can derive values of $\{w_{it} : i \in [N], t \in \mathcal{B}\}$, $\mathbf{d}$, $\mathbf{c}$ and $\mathbf{z}$ that yield a vector in $P_{\text{OCT-H}}$.

We first show that each vector in Q_{base} also belongs to $Q_{\text{OCT-H}}$. We need to verify that each vector in Q_{base} satisfies constraints (11). Let $i \in [N]$, $t \in \mathcal{B}$ and $\hat{t} \in \mathcal{L}(2t)$. By (12) we have

$$\begin{aligned} \mathbf{a}_t^\top \mathbf{x}^i &\leq b_t + M_i \left(1 - w_{i,\hat{t}}\right) - M_i \sum_{t' \in \mathcal{L}(2t) \setminus \{\hat{t}\}} w_{i,t'} \\ &\leq b_t + M_i' \left(1 - w_{i,\hat{t}}\right) \end{aligned}$$

and

$$\begin{aligned} \mathbf{a}_t^\top \mathbf{x}^i - \epsilon &\geq b_t - (M_i + \epsilon) \left(1 - w_{i,\hat{t}}\right) + (M_i + \epsilon) \sum_{t' \in \mathcal{L}(2t) \setminus \{\hat{t}\}} w_{i,t'} \\ &\geq b_t - (M_i' + \epsilon) \left(1 - w_{i,\hat{t}}\right), \end{aligned}$$

since $M_i \leq M_i' = 2$ for all $i \in N$ and by the non-negativity of $\mathbf{w}$.

To prove that the inclusion is strict, we exhibit a vector that is in $Q_{\text{OCT-H}}$ and not in Q_{base}. Consider a depth 2 tree and a training set consisting of two observations with identical feature vectors but different labels, e.g. $(x^1, y^1) = (1, 1)$ and $(x^2, y^2) = (1, 2)$. Note that $M_i = M_i' = 2$ for $i = 1, 2$. We define $w_{11} = w_{21} = 1$, $w_{12} = 1$, $w_{22} = 0$, $w_{13} = 0$, $w_{23} = 1$, $w_{14} = w_{15} = 1/2$, $w_{24} = w_{25} = 0$, $w_{16} = w_{17} = 0$, $w_{26} = w_{27} = 1/2$.Moreover, for each branching node $t \in \mathcal{B}$ we define $a_t = 1$ and $b_t = 0$. It can be easily checked that this vector is in $Q_{\text{OCT-H}}$. In particular, at each branching node $t \in \mathcal{B}$ the big-M constraints (11a) are $x \leq M_i'/2 = 1$, which is satisfied for $i = 1, 2$, while the big-M constraints (11b) are $x - \epsilon \geq -(M_i' + \epsilon)/2 = -1 - \epsilon/2$, which is satisfied for $i = 1, 2$ and $\epsilon \leq 4$.

On the other hand, some big-M constraints of the baseline formulation are violated, specifically at the root node the big-M constraint (6a) is $x^1 \leq 0$, which is violated since $x^1 = 1$. There is actually no hyperplane $a_1 x = b_1$ such that the big-M constraints (6) at $t = 1$ can be satisfied, since $x^1 = x^2$. □

We finally remark that our formulation uses fewer big-M constraints than OCT-H to enforce (1m). Upon enumerating for every branch node $t \in \mathcal{B}$, there are

$2N|\mathcal{B}| = N(2^{D+1} - 2)$ constraints among (6). In contrast, OCT-H (as well as OCT, the univariate decision tree formulation of [6]) requires $2N|\mathcal{L}|D = 2^{D+1}ND$ big-M constraints to accomplish the same goal of enforcing (1m). Since the number of constraints (1b)–(1h) is roughly $5N2^D$, the total number of constraints is reduced by a factor of 28%, 39%, 47% and 54% for depths 2, 3, 4 and 5, respectively.

We conclude that the use of the ancestor sets $\mathcal{A}_L(t)$ and $\mathcal{A}_R(t)$ weakens OCT-H. These ancestor sets are also used in OCT, as well as [16] and [13].

4 Shattering Inequalities

In this section, we propose a new class of valid inequalities for (1), called shattering inequalities, which correspond to subsets of observations that cannot be shattered by a multivariate split, and we propose a separation algorithm to generate these inequalities.

Let $\mathcal{C}$ be a family of binary classifiers in $\mathbb{R}^p$. A set of observations is *shattered* by $\mathcal{C}$ if, for any assignment of binary labels to these observations, there exists some classifier in $\mathcal{C}$ that can perfectly separate all the observations. The maximum number of observations that can be shattered by $\mathcal{C}$ is called the *Vapnik–Chervonenkis (VC) dimension* of $\mathcal{C}$ [33].

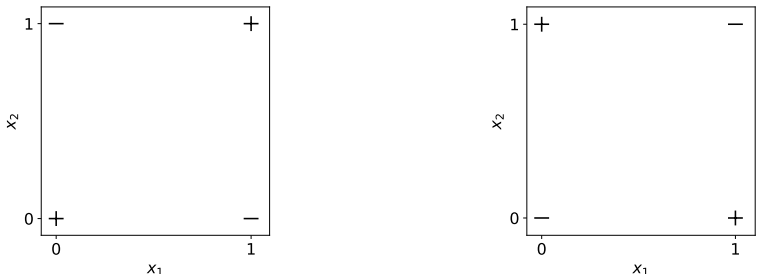
We now consider the family of binary classifiers $\mathcal{H}$ consisting of the multivariate splits in $\mathbb{R}^p$. Let $\mathcal{I}$ be a collection of subsets $I \subseteq [N]$ of observations such that $\{x^i\}_{i \in I}$ cannot be shattered by $\mathcal{H}$. For each $I \in \mathcal{I}$, denote by $\Lambda(I) \subset \{-1, 1\}^I$ the assignments of binary labels to observations in I so that they cannot be perfectly separated by any multivariate split in $\mathbb{R}^p$. Then, the following inequalities are valid for (1):

$$\sum_{i \in I : \lambda_i = -1} w_{i,2t} + \sum_{i \in I : \lambda_i = +1} w_{i,2t+1} \leq |I| - 1, \quad \forall I \in \mathcal{I},\ \lambda \in \Lambda(I),\ t \in \mathcal{B}. \tag{13}$$

The shattering inequalities (13) have the form of *packing constraints* [34] and impose the condition that at least one observation in I is *not* routed to the children of t as prescribed by the label assignment λ. We can restrict our attention to the minimal (w.r.t. set inclusion) subsets of $\mathcal{I}$. Indeed, if $I \in \mathcal{I}$ is *not* minimal, then each inequality (13) associated to I is implied by an inequality (13) associated to some $I' \subset I$ in $\mathcal{I}$.

Moreover, if I is a minimal set of observations in $\mathbb{R}^p$ that cannot be shattered by $\mathcal{H}$, then $|I| \leq p + 2$. This follows from the fact that the VC dimension of $\mathcal{H}$ is $p + 1$. Note that we might still be unable to perfectly split $|I| < p + 2$ observations in $\mathbb{R}^p$ if there exists an hyperplane that contains more than p points. For example, for $p = 2$, three points on a line labeled (in sequence) $1, -1, 1$ cannot be perfectly split. As a consequence, the support of inequalities (13) corresponding to minimal sets of observations in $\mathbb{R}^p$ that cannot be shattered by $\mathcal{H}$, is at most $p + 2$. In particular, if $p \ll N$, these inequalities are sparse.

Figure 2 shows an example using a dataset with points $\mathbf{x}^i = (x_1^i, x_2^i) \in \mathbb{R}^2$, where for the first four observations, $\mathbf{x}^1 = (0, 0)$, $\mathbf{x}^2 = (0, 1)$, $\mathbf{x}^3 = (1, 0)$, $\mathbf{x}^4 = (1, 1)$; the full dataset may contain many more observations. $I = \{1, 2, 3, 4\}$ is an example of a minimal subset of observations that cannot be shattered by $\mathcal{H}$; no hyperplane is



(a) (b)

Fig. 2 $I = \{1, 2, 3, 4\}$ is a minimal subset of observations that cannot be shattered by $\mathcal{H}$. $\Lambda(I)$ contains exactly two vectors $\lambda = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)$. (a) shows $\lambda = (+1, -1, -1, +1)$, which is used to derive (14); (b) shows $\lambda = (-1, +1, +1, -1)$, which is used to derive (15).

capable of separating $\{\mathbf{x}^1, \mathbf{x}^4\}$ from $\{\mathbf{x}^2, \mathbf{x}^3\}$, however $I \setminus \{i\}$ can be shattered for any $i \in I$. We can derive the shattering inequalities:

$$w_{2,2t} + w_{3,2t} + w_{1,2t+1} + w_{4,2t+1} \leq 3, \quad \forall t \in \mathcal{B} \tag{14}$$

and

$$w_{1,2t} + w_{4,2t} + w_{2,2t+1} + w_{3,2t+1} \leq 3, \quad \forall t \in \mathcal{B}. \tag{15}$$

4.1 Separation

It is impractical to enumerate all possible shattering inequalities (13) as there are exponentially many. Instead, they should be used sparingly, such as through row generation. Consider a vector $\bar{\mathbf{w}}$, possibly fractional, satisfying (1b) and (1c). We propose a method for separating $\bar{\mathbf{w}}$ using a violated inequality (13).

For each $t \in \mathcal{B}$, let $N_t(\bar{\mathbf{w}}) = \{i \in [N] : \bar{w}_{it} > \frac{p+1}{p+2}\}$; note that $N_{2t}(\bar{\mathbf{w}})$ and $N_{2t+1}(\bar{\mathbf{w}})$ are disjoint, since for all $i \in [N]$, at most one of $\bar{w}_{i,2t}, \bar{w}_{i,2t+1}$ can be greater than $\frac{p+1}{p+2}$. Consider the following system of linear inequalities in variables $(\mathbf{a}_t, b_t)$:

$$\begin{cases} \mathbf{a}_t^\top \mathbf{x}^i + 1 \leq b_t, & \forall i \in N_{2t}(\bar{\mathbf{w}}), \\ \mathbf{a}_t^\top \mathbf{x}^i - 1 \geq b_t, & \forall i \in N_{2t+1}(\bar{\mathbf{w}}). \end{cases} \tag{16}$$

Our goal is to either certify that system (16) is feasible, or return an inclusion-minimal subset of observations $I' \subseteq N_t(\bar{\mathbf{w}})$, such that $I' \cap N_{2t}(\bar{\mathbf{w}})$ cannot be perfectly separated from $I' \cap N_{2t+1}(\bar{\mathbf{w}})$ by a multivariate split. Each such subset I' corresponds to an *Irreducible Infeasible Subsystem* (IIS) of the infeasible system (16), which is defined as a subsystem of (16) that would become feasible by discarding one arbitrary inequality. Given an IIS of (16) indexed by I', the inequality:

$$\sum_{i \in I' \cap N_{2t}(\bar{\mathbf{w}})} w_{i,2t} + \sum_{i \in I' \cap N_{2t+1}(\bar{\mathbf{w}})} w_{i,2t+1} \leq |I'| - 1. \tag{17}$$

is clearly a shattering inequality (13). In the next lemma, we formally prove that inequality (17) is violated by $\bar{\mathbf{w}}$.

Proposition 2. *Let $\bar{\mathbf{w}}$ be a nonnegative vector satisfying (1b) and (1c) such that (16) is infeasible. Then, for each IIS I' of (16), the shattering inequality (17) is violated by $\bar{\mathbf{w}}$.*

Proof. Since $\bar{w}_{it} > \frac{p+1}{p+2}$ for all $i \in N_{2t}(\bar{\mathbf{w}}) \cup N_{2t+1}(\bar{\mathbf{w}})$, the right-hand-side of (17) evaluated in $\bar{\mathbf{w}}$ is strictly greater than

$$|I' \cap N_{2t}(\bar{\mathbf{w}})|\frac{p+1}{p+2} + |I' \cap N_{2t+1}(\bar{\mathbf{w}})|\frac{p+1}{p+2} = |I'|\frac{p+1}{p+2}.$$

Moreover

$$|I'|\frac{p+1}{p+2} \geq |I'| - 1 \iff |I'|\left(1 - \frac{p+1}{p+2}\right) \leq 1.$$

The condition on the right is always satisfied, since

$$|I'|\left(1 - \frac{p+1}{p+2}\right) = |I'|\frac{1}{p+2} \leq 1,$$

where the last inequality follows from the fact that $|I'| \leq p+2$. □

If $\bar{\mathbf{w}}$ is fractional and system (16) is feasible, we are not able to generate a shattering inequality violated by $\bar{\mathbf{w}}$, however there might exist one such violated inequality. On the other hand, if $\bar{\mathbf{w}}$ is binary and (16) is feasible we can conclude that $\bar{\mathbf{w}}$ satisfies all the shattering inequalities (13).

Proposition 3. *Let $\bar{\mathbf{w}}$ be a binary vector satisfying (1b) and (1c) and such that (16) is feasible. Then, every shattering inequality (13) defined at node t is satisfied by $\bar{\mathbf{w}}$.*

Proof. By contradiction, suppose that one shattering inequality (13) defined at node t is violated by $\bar{\mathbf{w}}$:

$$\sum_{i \in I : \lambda_i = -1} \bar{w}_{i,2t} + \sum_{i \in I : \lambda_i = +1} \bar{w}_{i,2t+1} > |I| - 1,$$

for some $I \in \mathcal{I}$ and $\lambda \in \Lambda(I)$. Since $\bar{\mathbf{w}}$ is binary, we must have that $\bar{w}_{i,2t} = 1$ for all $i \in I : \lambda_i = -1$ and $\bar{w}_{i,2t+1} = 1$ for all $i \in I : \lambda_i = 1$, thus $\{i \in I : \lambda_i = -1\} = I \cap N_{2t}(\bar{\mathbf{w}})$ and $\{i \in I : \lambda_i = 1\} = I \cap N_{2t+1}(\bar{\mathbf{w}})$. By the definition, each shattering inequality corresponds to a subset of observations that cannot be linearly separated, thus the system

$$\begin{cases} \mathbf{a}_t^\top \mathbf{x}^i + 1 \leq b_t, & \forall i \in I \cap N_{2t}(\bar{\mathbf{w}}), \\ \mathbf{a}_t^\top \mathbf{x}^i - 1 \geq b_t, & \forall i \in I \cap N_{2t+1}(\bar{\mathbf{w}}). \end{cases}$$

is infeasible. This is a subsystem of system (16), which contradicts that system (16) is feasible. □

Propositions 2 and 3 imply that, in the case where $\bar{\mathbf{w}}$ is binary, our separation algorithm is exact: either a violated inequality (17) is found, or $\bar{\mathbf{w}}$ satisfies the complicating constraint (1m) associated with node t. As we will discuss later, this implies

that when solving using lazy cuts, we can certify an integer incumbent solution as being optimal if we are unable to find a violated inequality (17).

Next, we discuss how to generate violated shattering inequalities. If system (16) is infeasible for a given $\bar{\mathbf{w}}$, to find an IIS of the system we apply Farkas' lemma and we construct a dual polyhedron $Q_t(\bar{\mathbf{w}})$ defined as

$$Q_t(\bar{\mathbf{w}}) = \Big\{ \mathbf{q} \in \mathbb{R}_+^{N_t(\bar{\mathbf{w}})} : \quad \sum_{i \in N_{2t}(\bar{\mathbf{w}})} q_i \mathbf{x}^i = \sum_{i \in N_{2t+1}(\bar{\mathbf{w}})} q_i \mathbf{x}^i, \\ \sum_{i \in N_{2t}(\bar{\mathbf{w}})} q_i = \sum_{i \in N_{2t+1}(\bar{\mathbf{w}})} q_i = 1 \Big\}.$$

In fact, there is a one-to-one correspondence between the IISs of (16) and the vertices of $Q_t(\bar{\mathbf{w}})$ [35]. Specifically, the indices of the inequalities appearing in an IIS of (16) correspond to the support of a vertex of $Q_t(\bar{\mathbf{w}})$, and vice versa. We remark that the polyhedron $Q_t(\bar{\mathbf{w}})$ has a very nice geometric interpretation. The decision variables $\mathbf{q}$ are associated with the observations indexed by $N_t(\bar{\mathbf{w}})$, and they can be interpreted as the coefficients of two convex combinations, one on the observations in $N_{2t}(\bar{\mathbf{w}})$ and the other on the observations in $N_{2t+1}(\bar{\mathbf{w}})$. It is evident from its definition that $Q_t(\bar{\mathbf{w}})$ is nonempty if and only if there exists a point that is both in the convex hull of $\{x^i : i \in N_{2t}(\bar{\mathbf{w}})\}$ and in the convex hull of $\{x^i : i \in N_{2t+1}(\bar{\mathbf{w}})\}$.

Based on the above discussion, we can define a separation algorithm to dynamically generate the shattering inequalities (13). This algorithm receives as input a vector $\bar{\mathbf{w}}$ and a fixed branch node $t \in \mathcal{B}$, and attempts to return an inequality of family (13) associated with node t that is violated by $\bar{\mathbf{w}}$. Precisely, the algorithm checks the feasibility of the dual polyhedron $Q_t(\bar{\mathbf{w}})$. If $Q_t(\bar{\mathbf{w}})$ is nonempty, then by Farkas' Lemma, system (16) is infeasible, and from each vertex of $Q_t(\bar{\mathbf{w}})$ we can construct an IIS of (16) and a corresponding shattering inequality (17) that is violated by $\bar{\mathbf{w}}$.

In practice, it is possible to efficiently generate multiple inequalities (17) by finding multiple vertices of $Q_t(\bar{\mathbf{w}})$. One method for finding multiple vertices is to optimize over $Q_t(\bar{\mathbf{w}})$ multiple times with different objective functions. For instance, let $\mathbf{f} \in \mathbb{Z}_+^{N_t(\bar{\mathbf{w}})}$ be a counter for the number of inequalities (17) that each observation in $N_t(\bar{\mathbf{w}})$ has appeared in thus far. One can repeatedly solve $\max\{\mathbf{f}^\top \mathbf{q} : \mathbf{q} \in Q_t(\bar{\mathbf{w}})\}$, each time updating a global counter $\mathbf{g} \in \mathbb{Z}_+^N$ as a cut is added. A downside to taking $\mathbf{f}$ to be a counter is that over time, $\mathbf{f}$ may converge to $\theta \mathbf{1}$ where $\theta \gg 0$, thus every observation is equally weighted in the objective. A refinement of this idea is to keep an exponential average; let $\mathbf{g} \in \mathbb{R}_+^N$ be initialized to the zero vector, and when a cut (17) is found, set $\mathbf{g} \coloneqq \mathbf{1}_{N_t(\bar{\mathbf{w}})} + 0.5\mathbf{g}$ where $\mathbf{1}_{N_t(\bar{\mathbf{w}})}$ is a binary vector with ones in indices corresponding to $N_t(\bar{\mathbf{w}})$. This objective function is a bounded vector (each component is at most 2), and places more emphasis on finding cuts not recently found.

Furthermore, for a fixed $\bar{\mathbf{w}}$, one can iterate t over a desired subset of branch nodes to generate even more cuts. Thus, the separation algorithm has the signature `Separation(`$\bar{\mathbf{w}}$`, nodes, n_cuts)`, where `nodes` is the desired subset of branch nodes

to loop over, and `n_cuts` is the number of times to repeatedly solve $\max\{\mathbf{f}^\top \mathbf{q} : \mathbf{q} \in Q_t(\bar{\mathbf{w}})\}$.

The separation algorithm can be implemented via LP with a run time that is polynomial in 2^D and size(X), where X is the $N \times p$ matrix encoding the features of the observations in the training set and size(X) is defined as the number of bits required to encode X [36]. Note that there is an exponential dependence with respect to the depth parameter D. However, the number of variables and constraints of our MIP formulation (as well as the other formulations from the literature) are already exponential in D and in practice, we want D to be small so that we can obtain a more interpretable decision tree.

We now propose three practical methods of using shattering inequalities as cuts. The first way is to add the cuts directly to the formulation up front as *initial cuts*. The advantage to adding cuts directly to the formulation as constraints rather than as lazy cuts or user cuts is that this enables the MIP solver to derive valid inequalities based on the initial cuts. To find such cuts, we solve the LP relaxation of (1) excluding the complicating constraints (1m). From the resulting solution, we call `Separation(`$\bar{\mathbf{w}}$`, branch_nodes, n_init_cuts)` in an attempt to derive cuts, where `n_init_cuts` is a user-defined parameter, and add all found cuts to the LP relaxation. We iterate this procedure until the LP relaxation amended with cuts no longer yields a solution from which cuts can be derived, or once 10 iterations have passed. Once this is done, we add to our baseline formulation all the derived cuts and solve the MIP.

The second way is to use the cuts as lazy cuts in a Benders-like decomposition. Without using big-M constraints (6), another option for implementing the complicating constraint (1m) is using shattering inequalities (13) exclusively. It is impractical to enumerate all possible shattering inequalities, however it might be the case that only a few such inequalities are needed to find an optimal solution. At nodes of the branch-and-cut tree, the MIP solver finds either a fractional solution of the relaxation or an (integer) incumbent solution $\bar{\mathbf{w}}$, thus giving the user an opportunity to add lazy cuts. Let `benders_nodes` be a user-defined subset of the branch nodes for which one would like to apply lazy cuts. For branch nodes not in `benders_nodes`, we add the associated big-M constraints (6) to the model (1). We call `Separation(`$\bar{\mathbf{w}}$`, benders_nodes, n_benders_cuts)` to add lazy cuts to (1), where `n_benders_cuts` is a user-defined parameter. In the case where we have an incumbent solution ($\bar{\mathbf{w}}$ is integer), separation is exact, meaning the solver correctly concludes the incumbent solution is feasible if no cuts are added. Thus, we can implement the complicating constraint (1m) associated with a branch node by dynamically adding shattering inequalities (13) to the model as lazy cuts.

The third way is to use the cuts as user cuts. User cuts are cutting planes added at nodes of the branch-and-cut tree in order to separate fractional solutions. Unlike lazy cuts which are required for model correctness, user cuts only serve to tighten the formulation; they cannot cut off integer solutions that satisfy the constraints of the master formulation (i.e., (1) minus the complicating constraints (1m)). Thus, for a given branch node, if one would like to apply user cuts, the big-M constraints (6) associated with that node must be present. Let `user_cuts_nodes` be a user-defined subset of the branch nodes for which one would like to apply user cuts; a requirement is that

`user_cuts_nodes` and `benders_nodes` must be disjoint. Note that `user_cuts_nodes` are a subset of the branch nodes not in `benders_nodes`, hence the big-M constraints (6) for `user_cuts_nodes` are in the model, as is required. At nodes of the branch-and-cut tree where the MIP solver has found a fractional solution $\bar{\mathbf{w}}$ of the relaxation, we call `Separation(`$\bar{\mathbf{w}}$`, user_cuts_nodes, n_user_cuts)` to add user cuts to (1), where `n_user_cuts` is a user-defined parameter.

We remark that in the second scenario, where we use the cuts as lazy cuts in a Benders-like decomposition, the shattering inequalities (17) are effectively *combinatorial Benders (CB) cuts* [37, 38]. CB cuts are a specialization of Hooker's logic-based Benders decomposition [38]. They are formally introduced by Codato and Fischetti [37], who study MIP problems that can be decomposed into a master problem with binary variables, and an LP subproblem whose feasibility depends from the solution of the master problem. Let $\mathcal{B}'$ be the subset of $\mathcal{B}$ containing the nodes in `benders_nodes`. To derive shattering inequalities as CB cuts, we can decompose (1) into a master problem and an LP feasibility subproblem as follows. The master problem is obtained from (1) by removing the complicating constraints (1m) for each $t \in \mathcal{B}'$ and by (possibly) adding some valid inequalities (13). Thus, the master problem is a MIP with decision variables $\mathbf{c}, \mathbf{d}, \mathbf{w}, \mathbf{z}$ and (a_t, b_t) for $t \in \mathcal{B} \setminus \mathcal{B}'$. The LP feasibility subproblem includes decision variables $(\mathbf{a}_t, b_t)$, $t \in \mathcal{B}'$, and verifies that, for a given $\bar{\mathbf{w}}$, $\mathcal{H}_t(\bar{\mathbf{w}})$ is a nonempty polyhedron for all $t \in \mathcal{B}'$. The shattering inequalities (17) can be interpreted as CB cuts by enforcing the complicating constraints (1m) through the following logical constraints (note that left children have an even index, while right children have an odd index):

$$\forall i \in [N],\ t \in \mathcal{B}' \cup \mathcal{L} \setminus \{1\},\ w_{it} = 1 \implies \begin{cases} \mathbf{a}_{t/2}^\top x^i + 1 \le b_{t/2} & \text{if } t \text{ is even} \\ \mathbf{a}_{\lfloor t/2 \rfloor}^\top x^i - 1 \ge b_{\lfloor t/2 \rfloor} & \text{if } t \text{ is odd.} \end{cases}$$

After solving the master problem, if the inequality system given by the activated logical constraints (i.e., those where $\bar{w}_{it} = 1$) is infeasible, the IISs of the system can be used to derive CB cuts. A key observation is that, in our setting, each IIS involves only the components of $\bar{\mathbf{w}}$ that pertain to a specific branch node $t \in \mathcal{B}'$. As a result, we can separately consider the inequality systems (16) associated with each individual branch node $t \in \mathcal{B}'$.

5 Heuristic for Training Multivariate Decision Trees

Unlike in the case of univariate decision trees, there are few widely used heuristic algorithms for training multivariate trees. Although this paper is primarily concerned with learning optimal decision trees, heuristics are still of interest as they serve as strong warm starts for the MIP model. In this section, we develop a novel heuristic based on existing univariate tree heuristics and support vector machines (SVMs). We take inspiration from [39], whose procedure uses a series of linear programs rather than SVMs.

We first review univariate decision tree heuristics in order to better understand the difficulty in extending these algorithms to the multivariate setting. Many univariate tree heuristics are top-down greedy algorithms that recursively split the data according to some optimization criterion. For example, the CART algorithm seeks the split resulting in the largest reduction in Gini impurity [1] The ID3, C4.5, and C5.0 algorithms function similarly with respect to Shannon entropy [5]. This optimization problem is easy to solve in the univariate setting as all $O(Np)$ possible splits may be enumerated, however such enumeration is not possible in the multivariate setting; indeed, the problem of finding a multivariate split that optimizes some arbitrary criterion such as Gini impurity or Shannon entropy is NP-hard [40]. Hence, it is impractical to simply replicate these univariate heuristics when allowing for multivariate splits.

Rather than enumerating all possible multivariate splits, we consider a small set of promising splits. In particular, we use the one-vs-one and one-vs-rest decision boundaries returned by linear SVMs as our set of candidate splits. Among these candidates, we pick the one that optimizes the desired split criterion. For example, if the criterion is Gini impurity, we pick the split that results in the greatest decrease in impurity. We build the tree in the same recursive fashion as top-down greedy algorithms for univariate trees. In the case where there is a limit $\beta < |\mathcal{B}|$ on the number of non-trivial splits used, we prune the tree.

Our heuristic, specifically the enumeration of SVM decision boundaries, can be implemented using scikit-learn's `LinearSVC` and `SVC(kernel='linear')` [41]. Multiple SVMs can be constructed by varying hyperparameters; we vary the `C` hyperparameter used in both scikit-learn models. Possible choices of minimization criteria include Gini impurity, entropy, and training error. We train three multivariate trees each based on one of these criteria, and pick the one with the highest overall training accuracy to be the warm start.

6 Experiments

In this section, we provide two sets of numerical experiments to benchmark various implementations of our approach (S-OCT) with five methods from the literature: OCT and OCT-H [6], two MIP models for learning optimal univariate and multivariate trees respectively; FlowOCT [8], a MIP model which can be solved as-is (FlowOCT), or with Benders decomposition (FlowOCT-Benders); and DL8.5 [25], an itemset mining-based approach that uses branch-and-bound and caching.

6.1 Experimental Setup

Before comparing S-OCT against the other approaches from the literature, we tuned S-OCT on five datasets to find four promising configurations of the S-OCT model parameters `n_init_cuts`, `benders_nodes`, `n_benders_cuts`, `user_cuts_nodes`, and `n_user_cuts`. The five datasets we used can be obtained from the UCI Machine Learning Repository [42]: Hayes-Roth, Tic-Tac-Toe Endgame, Climate Model Crashes, Glass Identification and Image Segmentation. We use the four best performing versions of our model, along with the baseline S-OCT model which does not use any shattering inequalities, for the remaining experiments.

We benchmark these five variants of our approach against other methods from the literature on the following ten datasets from the UCI Machine Learning Repository [42]: (A) Balance Scale, (B) Congressional Voting Records, (C) Soybean (Small), (D) Iris, (E)Wine, (F) Breast Cancer, (G) Banknote Authentication, (H) Blood Transfusion, (I) Ionosphere, and (J) Parkinsons. Note that none of these datasets were used to tune S-OCT. The first set of benchmarks is a direct MIP comparison, where we compare the five S-OCT variants with the following MIP methods: OCT, OCT-H, FlowOCT, and FlowOCT-Benders. The second set of benchmarks is a comprehensive comparison where we compare against all of the previously mentioned methods: OCT, OCT-H, FlowOCT, FlowOCT-Benders, and DL8.5. In the direct MIP comparison, we use our own implementations of OCT and OCT-H. In the comprehensive comparison, we use the implementations of OCT and OCT-H from the Interpretable AI package [43]. In both the direct MIP and comprehensive comparisons, we use our own implementations of FlowOCT and FlowOCT-Benders. We use the DL8.5 package available on PyPI [26][3].

Since FlowOCT and DL8.5 require binary features, for these models we perform an additional *bucketization* step for the numerical datasets. For FlowOCT, we define five bins per feature using quantiles. For DL8.5, we perform bucketization as follows: for every feature j, we sort the observations according to this feature, find consecutive observations with different class labels (and different values for feature j), and define a binary feature that has value 1 if and only if x_j is less than the average of the two adjacent feature values. Note that both methods of bucketizing continuous data may sacrifice training accuracy, as identified in [28]. For OCT, OCT-H, and S-OCT, we normalize numerical features to the $[0, 1]$ interval. For all models, we perform one-hot encoding for categorical features.

Our experiments were programmed using Python 3.8.10 and all optimization problems were solved using Gurobi 10.0 [44] on a machine with a 3.00 GHz 6-core Intel Core i5-8500 processor and 16 GB RAM. A 10-minute time limit was imposed for all optimization problems. Our code can be found at https://github.com/zachzhou777/S-OCT.

6.1.1 Tuning experiments.

Let $\mathcal{B}_{\text{last}} = \{2^{D-1}, \ldots, 2^D - 1\}$ denote the last level of branch nodes in the tree. We tried three general variations of our S-OCT model, each associated with the three different methods of using our cuts:

- **Initial cuts.** Let `SOCT-init-<n_init_cuts>` denote the baseline S-OCT model amended with initial cuts, with cuts being found using the procedure `Separation(`$\bar{\mathbf{w}}$`, branch_nodes, n_init_cuts)`. We defined the models `SOCT-init-1` and `SOCT-init-5`.
- **Benders/lazy cuts.** Let `SOCT-benders-<benders_nodes>-<n_benders_cuts>` denote the S-OCT model where shattering inequalities (13) are used to implement the complicating constraints (1m) for branch nodes

[3]The package is now named pydl8.5, not dl8.5

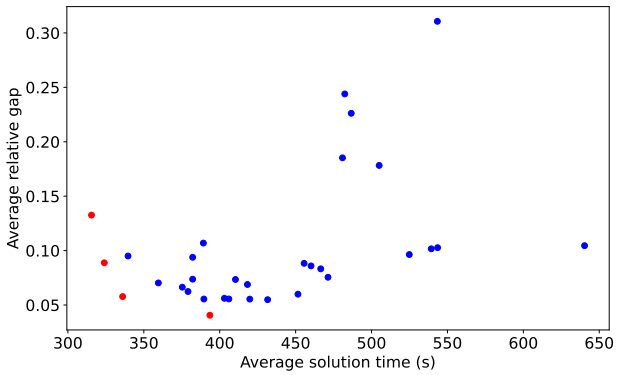


Fig. 3 A scatter plot of the average solution time and average relative gap for all parameter tuning combinations. The red dots indicate the four best performing implementations.

in the set `benders_nodes`, with cuts being found using the procedure `Separation(`$\bar{\mathbf{w}}$`, benders_nodes, n_benders_cuts)`. We defined the models `SOCT-benders-<benders_nodes>-<n_benders_cuts>` for $(\texttt{benders\_nodes}, \texttt{n\_benders\_cuts}) \in \{\{1\}, \mathcal{B}_{\text{last}}, \mathcal{B}\} \times \{1, 5, 10, 50, 100\}$ (i.e., the options for `benders_nodes`are the root node only, the last level of branch nodes, and the set of all branch nodes).

- **User cuts.** Let `SOCT-user-<user_cuts_nodes>-<n_user_cuts>` denote the baseline S-OCT model where user cuts are found for branch nodes in the set `user_cuts_nodes`, with cuts being found using the procedure `Separation(`$\bar{\mathbf{w}}$`, user_cuts_nodes, n_user_cuts)`. We defined the models `SOCT-user-<user_cuts_nodes>-<n_user_cuts>` for $(\texttt{user\_cuts\_nodes}, \texttt{n\_user\_cuts}) \in \{\{1\}, \mathcal{B}_{\text{last}}, \mathcal{B}\} \times \{1, 5, 10, 50, 100\}$.

For each variation of our model, we simply run it across the five datasets. No cross validation is performed. We do not employ warm starts or regularization (constraint (1h) is deleted). Figure 3 displays a scatter plot of the average solution time and average relative gap for all parameter tuning implementations across depths 2,3 and 4. We selected the four best performing implementations representing the pareto-frontier (shown in red in Figure 3), along with the baseline model: `SOCT-baseline`, `SOCT-benders-last-1`, `SOCT-benders-last-10`, `SOCT-init-1`, `SOCT-init-5`.

6.1.2 Direct MIP comparison.

In the direct MIP comparison, we compare the performance of our five S-OCT variants against the other MIP methods: OCT, OCT-H, FlowOCT, and FlowOCT-Benders. Here, we are interested in the strength of the above MIP formulations, rather than their efficacy as machine learning models. Thus, we do not include constraint (1h) in S-OCT and we set $\alpha = 0$ in the other models. In other words, we simply maximize the number of training points correctly classified. Finally, we refrain from providing warm starts. We recall that OCT, FlowOCT, and FlowOCT-Benders solve the optimal *univariate* decision tree problem, while OCT-H and our five S-OCT variants solve the optimal *multivariate* decision tree problem. These two problems are inherently different, thus

when comparing the performance of a univariate and a multivariate model we should mainly focus on the trade-off between training time and training accuracy. For all MIP models, we set the Gurobi relative gap parameter `MIPGap` to 0 and the absolute gap parameter `MIPGapAbs` to 1; this is done to prevent the solver from terminating before the time limit unless it has a truly optimal solution, as by default Gurobi terminates when the relative gap is below 10^{-4}. For each instance (dataset, depth D), we train each model on the entire dataset (no train-test split or cross validation) and record the following:

- Training time, i.e., solution time.
- Upper bound: The bound on the best possible objective. Also known as the dual objective bound and denoted by z_D.
- Lower bound: The objective of the best incumbent at termination. Also known as the primal objective bound and denoted by z_P.
- Relative MIP gap: For a maximization problem, $gap = (z_D - z_P)/z_P$; if $z_P = z_D = 0$, then $gap = 0$.

Figure 4(a) displays boxplots (across the ten datasets) of solution time for each model and each depth. At depth 2, FlowOCT-Benders performed best with an average±standard deviation solution time of 15.93 ± 21.01s. The fastest multivariate model was OCT-H, with an average solution time of 88.4 ± 173.91. Our best performing model was SOCT-Benders-last-10 with an average solution time of 95.15 ± 183.07s, following closely by SOCT-Benders-last-1 with an average solution time of 97.35 ± 178.54s. At depth 3, SOCT-Benders-last-1 and SOCT-Benders-last-10 performed best with an average solution time of 72.23 ± 178.72s and 111.71 ± 188.12s, respectively. SOCT-Benders-last-1 improved upon the best model from the literature (OCT-H) by 184.44s (72%). The best univariate model is FlowOCT, with an average solution time of 343.23±261.78s. At depth 4, SOCT-Benders-last-1 and SOCT-Benders-last-10 performed best with an average solution time of 173.91 ± 215.82s and 150.15 ± 247.61s, respectively. SOCT-Benders-last-10 improved upon the best model from the literature (OCT-H) by 151.12s (50%). The best univariate model is OCT, with an average solution time of 483.43 ± 236.87s.

Figure 4(b) displays boxplots (across the ten datasets) of relative gap for each model and each depth. At depths 2 and 3, the univariate model FlowOCT performed best with an average relative gap of 0 ± 0 and 0.02 ± 0.03, respectively. The best multivariate models at depth 2 are SOCT-baseline, SOCT-init-1 and SOCT-init-5, with an average relative gap of 0.03 ± 0.08, closely followed by OCT-H, with an average relative gap of 0.03 ± 0.08. SOCT-benders-last-1 and SOCT-benders-last-10 produced an average relative gap of 0.03 ± 0.09 for all three depths, improving over OCT-H by 59% and 64% at depths 3 and 4, respectively and over FlowOCT-Benders, which is the best univariate model at depth 4, by 50%.

Figure 4(c) displays boxplots (across the ten datasets) of in-sample accuracy for each model and each depth. SOCT-benders-last-1 performed best across all three depths with an average in-sample accuracy of 0.98 ± 0.07. The best performing model from the literature was OCT-H with an average in-sample accuracy of 0.95 ± 0.1.

Figure 4(d) displays a scatter plot of the overall average relative gap and overall average solution time across all ten datasets and three depths for each model. Overall, SOCT-benders-last-1 and SOCT-benders-last-10 performed best with an average solution time and relative gap of (114.5 ± 196.64s, 0.03 ± 0.09) and (119.0 ± 209.61s, 0.03 ± 0.09), respectively. The best performing models from the literature were FlowOCT and OCT-H with an average solution time and relative gap of (306.26 ± 279.87s, 0.03 ± 0.07) and (215.45 ± 248.35s, 0.06 ± 0.14), respectively.

6.1.3 Comprehensive comparison.

In the comprehensive comparison, we compare the practical performance of our approach against a wider range of other methods, both within and outside of MIP. We now allow the use of warm starts for the MIP models: FlowOCT and FlowOCT-Benders use scikit-learn's CART implementation [41], and our five SOCT models use the multivariate heuristic described in Section 5.

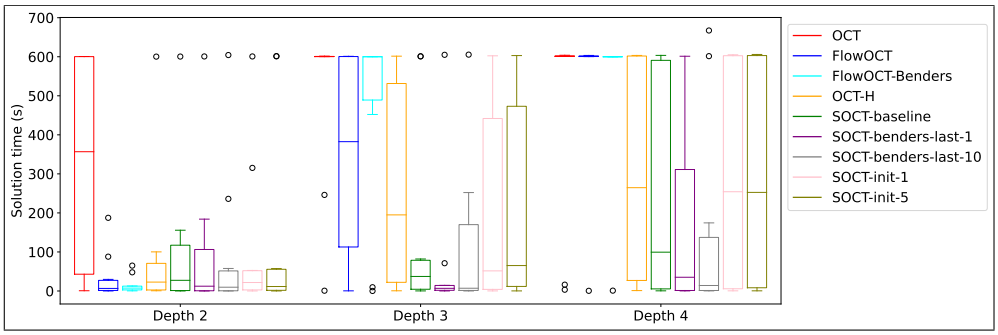
For each instance (dataset, depth D), we perform 3-fold cross validation. We tune hyperparameters using an inner 2-fold cross validation. For the S-OCT and FlowOCT models, we tuned the hyperparameter β, the maximum number of splits allowed in the tree; when $D = 2$ we searched over $\{2, 3\}$, when $D = 3$ we searched over $\{3, 5, 7\}$, when $D = 4$ we searched over $\{5, 10, 15\}$. Interpretable AI automatically tunes α for OCT and OCT-H. For DL8.5, we tuned the hyperparameter `min_sup`, which refers to the minimum number of observations per leaf; we searched over $\{1, 10, 20\}$. For each model and instance, we record for each of the three folds the following:

- Training time
- Train accuracy
- Test accuracy

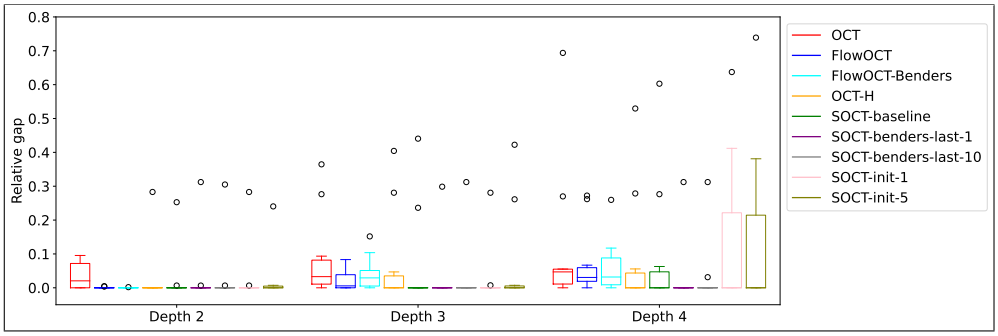
Tables 1, 2, and 3 provide a detailed comparison of the numerical results for each dataset and all model implementations at depths 2, 3, and 4, respectively. Figure 5 displays line plots of the training accuracy, testing accuracy, and solution time across all 10 datasets (sorted for clarity) and for all models at depth four. The line plots visualize the results in Table 3.

For solution time, our models performed similar to OCT and better than OCT-H, except for dataset H, where we consistently timed-out. Note that we are unable to determine if OCT and OCT-H solve to optimality or if the interpretable AI implementation uses an early stopping criteria. Our models to not use early stopping and unless they timeout, solve to optimality. Excluding dataset H, the average solution time for OCT at depths 2, 3, and 4 was 0.31 ± 0.09, 0.38 ± 0.16, and 0.44 ± 0.2, respectively. The average solution time for OCT-H at depths 2, 3, and 4 was 9.8 ± 8.4, 11.6 ± 9.9, and 12.2 ± 10.7, respectively. The average solution time for SOCT-benders-last-1 at depths 2, 3, and 4 was 0.71 ± 0.41, 1.4 ± 1.1, and 3.5 ± 5.8, respectively.

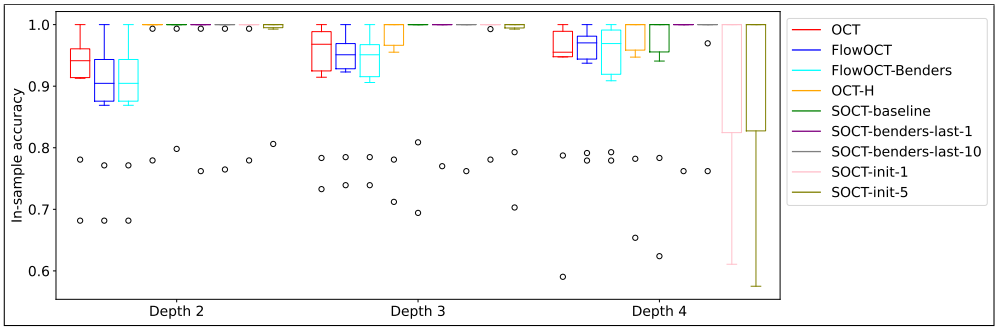
In terms of out-of-sample accuracy, our models performed similarly to OCT-H and significantly better than the other models from the literature. The average out-of-sample accuracy for OCT-H at depths 2, 3, and 4 was 0.92 ± 0.06, 0.93 ± 0.05, and 0.93 ± 0.06, respectively. The average solution time for SOCT-benders-last-1 at depths 2, 3, and 4 was 0.92 ± 0.07, 0.92 ± 0.08, and 0.93 ± 0.07, respectively.



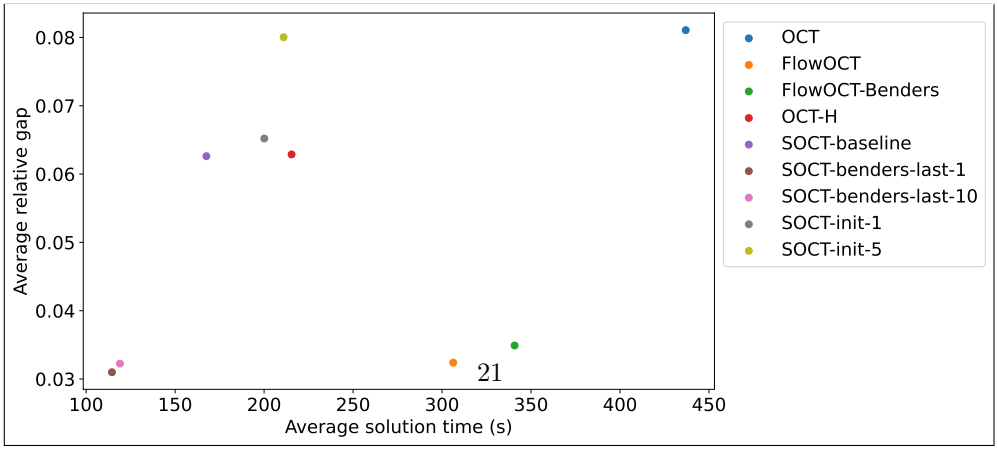
(a)



(b)



(c)



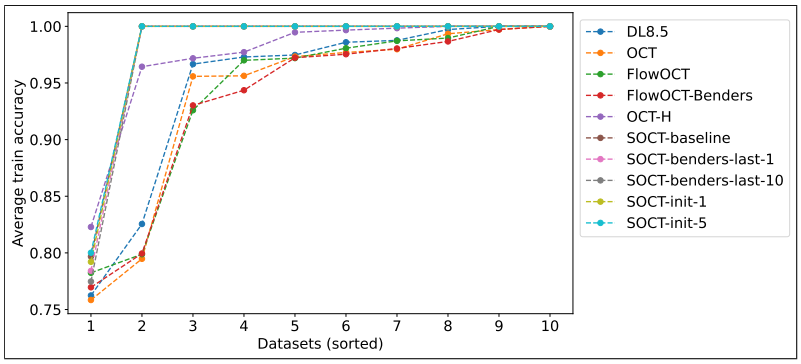
(d)

Fig. 4 Direct MIP comparison results displaying (a) boxplots (across datasets) of solution time for all three depths, (b) boxplots (across datasets) of relative gap for all three depths, (c) boxplots (across datasets) of in-sample accuracy for all three depths, and (d) the overall average relative gap and overall average solution time across all three depths and all datasets.

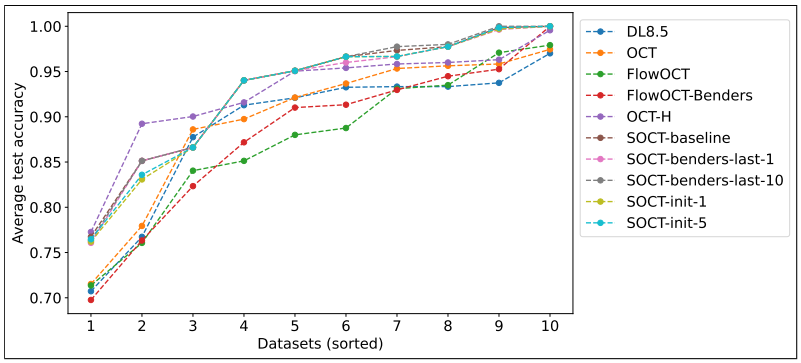
Overall, our models were able perform similarly to the best models from the literature in terms of out-of-sample accuracy, while improving solution time by between 71% and 93%.

7 Conclusion

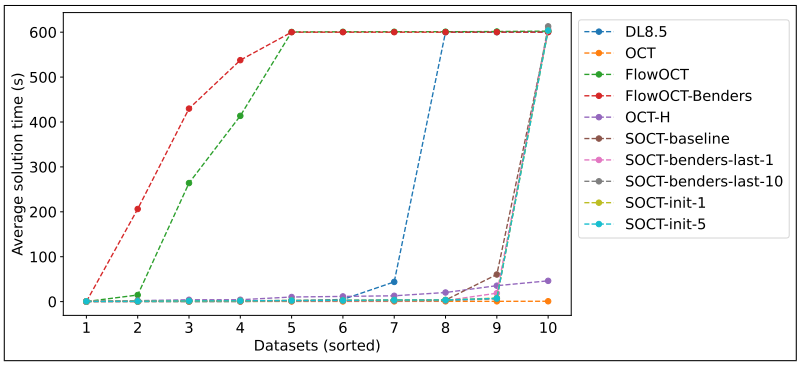
We proposed a new MIP formulation for the optimal decision tree problem. Our approach directly deals with numerical features and leverages the higher modeling power of multivariate branching rules. We also introduced a new class of valid inequalities and an exact decomposition approach that uses these inequalities as feasibility cuts. These inequalities exploit the structure of decision trees and express the geometrical properties of the dataset at hand. We demonstrate through numerical experiments that our MIP approach outperforms (in terms of training accuracy, testing accuracy, solution time, and relative gap) two other popular MIP formulations, and is able to improve solution time, while remaining competitive in terms of out-of-sample accuracy to a wide range of popular approaches from the literature. Finally, we note that our formulation and the shattering inequalities (13) are general and can be extended to any binary classifier used to implement the branching rules. When the branching rules are implemented via multivariate splits, the separation of the shattering inequalities can be performed efficiently. However, the separation may become more challenging if we consider more complex classifiers.



(a)



(b)



(c)

Fig. 5 Line plots across all 10 datasets (sorted for clarity) of the training accuracy, testing accuracy, and solution time for all models at depth four.

Table 1 Detailed summary of comprehensive comparisons for depth 2.

	Dataset									
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)	(J)
Observations (N)	625	435	47	150	178	569	1372	748	351	195
Features (p)	20	48	59	4	13	30	4	4	34	22
DL8.5 buckets (avg)	N/A	N/A	N/A	35.67	432.33	3143.67	1184.0	103.0	1517.0	758.0
FlowOCT buckets (avg)	N/A	N/A	N/A	20.0	65.0	150.0	20.0	18.67	158.0	110.0
Classes (K)	3	2	4	3	3	2	2	2	2	2
	In-sample accuracy									
DL8.5	0.7	0.96	1.0	0.96	0.97	0.97	0.93	0.78	0.92	0.94
OCT	0.69	0.96	1.0	0.97	0.96	0.95	0.93	0.76	0.92	0.89
FlowOCT	0.69	0.96	1.0	0.9	0.93	0.95	0.9	0.77	0.87	0.9
FlowOCT-Benders	0.69	0.96	0.94	0.9	0.93	0.95	0.9	0.77	0.87	0.9
OCT-H	1.0	0.96	0.99	0.99	1.0	0.99	1.0	0.82	0.96	0.95
SOCT-baseline	1.0	1.0	1.0	0.99	1.0	1.0	1.0	0.8	1.0	1.0
SOCT-benders-last-1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.79	1.0	1.0
SOCT-benders-last-10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0
SOCT-init-1	1.0	1.0	1.0	0.99	1.0	1.0	1.0	0.8	1.0	1.0
SOCT-init-5	1.0	1.0	1.0	0.99	1.0	1.0	1.0	0.79	1.0	1.0
	Out-of-sample accuracy									
DL8.5	0.63	0.95	0.98	0.95	0.94	0.93	0.91	0.76	0.86	0.85
OCT	0.66	0.96	0.96	0.93	0.9	0.92	0.91	0.76	0.86	0.85
FlowOCT	0.66	0.95	1.0	0.87	0.9	0.92	0.87	0.75	0.84	0.84
FlowOCT-Benders	0.66	0.95	0.92	0.87	0.9	0.92	0.87	0.75	0.84	0.87
OCT-H	0.92	0.96	0.96	0.94	0.91	0.96	1.0	0.78	0.89	0.88
SOCT-baseline	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.78	0.88	0.87
SOCT-benders-last-1	0.97	0.94	1.0	0.96	0.98	0.95	0.99	0.77	0.88	0.79
SOCT-benders-last-10	0.97	0.94	1.0	0.96	0.98	0.95	1.0	0.76	0.87	0.81
SOCT-init-1	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.78	0.88	0.83
SOCT-init-5	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.76	0.89	0.87
	Computational time (s)									
DL8.5	0.0	0.0	0.0	0.0	0.02	1.37	0.3	0.0	0.34	0.09
OCT	0.4	0.34	0.23	0.2	0.24	0.47	0.34	0.23	0.35	0.25
FlowOCT	3.73	5.45	0.04	0.15	0.97	64.13	2.74	0.83	32.15	7.45
FlowOCT-Benders	0.68	3.51	0.1	0.15	2.35	28.68	3.56	1.0	30.05	6.44
OCT-H	18.49	9.59	1.81	0.49	3.8	26.37	12.36	3.14	19.44	5.52
SOCT-baseline	1.38	0.59	0.28	0.5	0.33	1.59	11.84	601.31	2.35	4.53
SOCT-benders-last-1	1.05	0.5	0.26	0.5	0.29	0.78	1.58	601.38	1.35	0.68
SOCT-benders-last-10	1.06	0.53	0.26	0.53	0.3	0.77	4.15	601.42	1.06	0.66
SOCT-init-1	1.46	0.64	0.29	0.52	0.34	1.6	8.58	601.42	3.53	2.4
SOCT-init-5	1.46	0.66	0.29	0.51	0.34	1.53	44.34	601.74	5.63	2.29

Table 2 Detailed summary of comprehensive comparisons for depth 3.

	Dataset									
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)	(J)
Observations (N)	625	435	47	150	178	569	1372	748	351	195
Features (p)	20	48	59	4	13	30	4	4	34	22
DL8.5 buckets (avg)	N/A	N/A	N/A	35.67	432.33	3143.67	1184.0	103.0	1517.0	758.0
FlowOCT buckets (avg)	N/A	N/A	N/A	20.0	65.0	150.0	20.0	18.67	158.0	110.0
Classes (K)	3	2	4	3	3	2	2	2	2	2
	In-sample accuracy									
DL8.5	0.75	0.97	1.0	0.98	1.0	0.99	0.99	0.81	0.96	0.99
OCT	0.75	0.96	1.0	0.97	1.0	0.98	0.97	0.79	0.93	0.97
FlowOCT	0.74	0.97	1.0	0.94	0.98	0.96	0.95	0.79	0.91	0.94
FlowOCT-Benders	0.74	0.97	1.0	0.92	0.97	0.96	0.95	0.79	0.91	0.95
OCT-H	1.0	0.96	1.0	1.0	1.0	1.0	1.0	0.81	0.94	0.99
SOCT-baseline	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0
SOCT-benders-last-1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0
SOCT-benders-last-10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0
SOCT-init-1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0
SOCT-init-5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0
	Out-of-sample accuracy									
DL8.5	0.7	0.95	0.98	0.95	0.91	0.93	0.97	0.77	0.85	0.87
OCT	0.7	0.95	0.96	0.93	0.92	0.94	0.96	0.77	0.86	0.9
FlowOCT	0.69	0.94	0.98	0.89	0.92	0.94	0.93	0.76	0.85	0.86
FlowOCT-Benders	0.69	0.94	0.98	0.89	0.89	0.92	0.93	0.76	0.86	0.85
OCT-H	0.93	0.96	0.98	0.96	0.92	0.95	1.0	0.8	0.89	0.91
SOCT-baseline	0.97	0.94	1.0	0.96	0.98	0.95	1.0	0.76	0.87	0.87
SOCT-benders-last-1	0.97	0.94	1.0	0.96	0.98	0.95	0.99	0.76	0.86	0.81
SOCT-benders-last-10	0.97	0.94	1.0	0.96	0.98	0.95	1.0	0.76	0.86	0.77
SOCT-init-1	0.97	0.94	1.0	0.96	0.98	0.95	1.0	0.77	0.87	0.84
SOCT-init-5	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.76	0.87	0.85
	Computational time (s)									
DL8.5	0.0	0.01	0.0	0.0	0.46	600.68	111.65	0.1	192.8	12.67
OCT	0.41	0.41	0.21	0.2	0.26	0.71	0.51	0.28	0.5	0.3
FlowOCT	196.92	160.14	0.09	2.4	185.62	513.1	166.13	23.38	469.77	303.39
FlowOCT-Benders	105.99	82.1	0.01	1.81	111.88	600.04	505.68	224.65	429.5	600.03
OCT-H	22.58	10.4	2.16	0.62	4.16	31.52	13.15	6.24	26.33	8.28
SOCT-baseline	2.03	1.22	0.44	0.81	0.5	1.76	51.04	602.05	1.76	5.51
SOCT-benders-last-1	1.52	0.78	0.36	0.89	0.39	1.05	1.94	606.03	1.29	3.99
SOCT-benders-last-10	1.49	0.77	0.36	0.59	0.4	1.11	1.74	604.14	1.29	4.62
SOCT-init-1	2.23	1.24	0.44	1.0	0.52	1.86	6.71	602.27	1.83	5.17
SOCT-init-5	2.19	1.24	0.44	0.74	0.52	1.97	13.86	602.5	1.94	5.05

Table 3 Detailed summary of comprehensive comparisons for depth 4.

	Dataset									
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)	(J)
Observations (N)	625	435	47	150	178	569	1372	748	351	195
Features (p)	20	48	59	4	13	30	4	4	34	22
DL8.5 buckets (avg)	N/A	N/A	N/A	35.67	432.33	3143.67	1184.0	103.0	1517.0	758.0
FlowOCT buckets (avg)	N/A	N/A	N/A	20.0	65.0	150.0	20.0	18.67	158.0	110.0
Classes (K)	3	2	4	3	3	2	2	2	2	2
	In-sample accuracy									
DL8.5	0.76	0.97	1.0	0.97	1.0	0.99	0.99	0.83	0.97	1.0
OCT	0.76	0.96	1.0	0.97	1.0	0.98	0.99	0.79	0.96	0.98
FlowOCT	0.78	0.99	1.0	0.97	1.0	0.98	0.97	0.8	0.93	0.99
FlowOCT-Benders	0.77	0.98	1.0	0.99	1.0	0.98	0.97	0.8	0.93	0.94
OCT-H	1.0	0.96	1.0	1.0	1.0	0.99	1.0	0.82	0.98	0.97
SOCT-baseline	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0
SOCT-benders-last-1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0
SOCT-benders-last-10	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.77	1.0	1.0
SOCT-init-1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.79	1.0	1.0
SOCT-init-5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0
	Out-of-sample accuracy									
DL8.5	0.71	0.93	0.94	0.93	0.93	0.92	0.97	0.77	0.88	0.91
OCT	0.72	0.96	0.96	0.95	0.92	0.94	0.97	0.78	0.89	0.9
FlowOCT	0.71	0.93	0.98	0.88	0.89	0.93	0.97	0.76	0.84	0.85
FlowOCT-Benders	0.7	0.94	1.0	0.91	0.91	0.93	0.95	0.76	0.82	0.87
OCT-H	0.95	0.95	0.96	0.96	0.92	0.96	1.0	0.77	0.9	0.89
SOCT-baseline	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.77	0.87	0.85
SOCT-benders-last-1	0.97	0.94	1.0	0.96	0.98	0.95	1.0	0.76	0.87	0.85
SOCT-benders-last-10	0.97	0.94	1.0	0.98	0.98	0.95	1.0	0.76	0.87	0.85
SOCT-init-1	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.76	0.87	0.83
SOCT-init-5	0.97	0.94	1.0	0.97	0.98	0.95	1.0	0.76	0.87	0.84
	Computational time (s)									
DL8.5	0.03	0.15	0.0	0.01	43.91	600.6	600.11	4.88	600.12	1.61
OCT	0.54	0.45	0.21	0.2	0.29	0.82	0.65	0.34	0.62	0.36
FlowOCT	601.04	600.81	0.19	413.69	14.71	601.47	602.34	264.01	600.91	600.45
FlowOCT-Benders	537.69	429.82	0.01	600.03	206.25	600.05	600.04	600.04	600.05	600.03
OCT-H	20.3	11.6	2.21	0.66	4.13	35.49	12.98	4.2	46.27	10.26
SOCT-baseline	3.2	2.51	0.58	1.32	0.81	3.44	60.38	603.04	2.75	1.55
SOCT-benders-last-1	2.39	1.39	0.45	1.04	0.59	1.99	18.74	607.34	1.83	1.18
SOCT-benders-last-10	2.37	1.39	0.44	0.92	0.6	1.96	6.21	613.0	1.81	1.2
SOCT-init-1	3.5	2.55	0.6	1.3	0.83	3.65	8.01	603.56	2.81	1.82
SOCT-init-5	3.61	2.55	0.6	1.58	0.84	3.68	7.25	603.81	2.89	1.84

References

[1] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. Taylor & Francis, New York (1984)

[2] Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)

[3] Liaw, A., Wiener, M., *et al.*: Classification and regression by randomforest. R news **2**(3), 18–22 (2002)

[4] Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is np-complete. Information processing letters **5**(1), 15–17 (1976)

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

[6] Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning **106** (2017)

[7] Aghaei, S., Azizi, M.J., Vayanos, P.: Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making (2019)

[8] Aghaei, S., Gómez, A., Vayanos, P.: Strong Optimal Classification Trees. Optimization Online (2021). https://optimization-online.org/?p=16925

[9] Aghaei, S., Gómez, A., Jo, N., Vayanos, P.: Learning Optimal Prescriptive Trees from Observational Data. Optimization Online (2021). https://optimization-online.org/?p=17313

[10] Justin, N., Aghaei, S., Gómez, A., Vayanos, P.: Optimal robust classification trees. In: The AAAI-22 Workshop on Adversarial Machine Learning and Beyond (2022). https://openreview.net/forum?id=HbasA9ysA3

[11] Jo, N., Aghaei, S., Benson, J., Gómez, A., Vayanos, P.: Learning optimal fair classification trees. CoRR **abs/2201.09932** (2022) 2201.09932

[12] Dash, S., Günlük, O., Wei, D.: Boolean Decision Rules via Column Generation (2020)

[13] Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., Scheinberg, K.: Optimal Generalized Decision Trees via Integer Programming (2019)

[14] Verwer, S., Zhang, Y.: Learning decision trees with flexible constraints and objectives using integer optimization. In: Salvagnin, D., Lombardi, M. (eds.) Integration of AI and OR Techniques in Constraint Programming, pp. 94–103. Springer, Cham (2017)

[15] Verwer, S., Zhang, Y.: Learning optimal classification trees using a binary linear program formulation. In: Proceedings of the Thirty-Third AAAI Conference on

Artificial Intelligence (AAAI-19), pp. 1625–1632. AAAI Press, ??? (2019). 33rd AAAI Conference on Artificial Intelligence, AAAI-19 ; Conference date: 27-01-2019 Through 01-02-2019

[16] Zhu, H., Murali, P., Phan, D.T., Nguyen, L.M., Kalagnanam, J.: A scalable mip-based method for learning optimal multivariate decision trees. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual (2020)

[17] Wolsey, L.A.: Integer Programming. Wiley Series in Discrete Mathematics and Optimization. Wiley, Hoboken, New Jersey, U.S. (1998)

[18] Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J.: Learning optimal decision trees with sat. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. IJCAI'18, pp. 1362–1368. AAAI Press, Stockholm, Sweden (2018)

[19] Avellaneda, F.: Efficient inference of optimal decision trees. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 3195–3202 (2020)

[20] Janota, M., Morgado, A.: Sat-based encodings for optimal decision trees with explicit paths. In: Pulina, L., Seidl, M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2020, pp. 501–518. Springer, Cham (2020)

[21] Schidler, A., Szeider, S.: Sat-based decision tree learning for large data sets. Proceedings of the AAAI Conference on Artificial Intelligence **35**(5), 3904–3912 (2021)

[22] Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., Schaus, P.: Learning optimal decision trees using constraint programming (extended abstract). In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pp. 4765–4769. International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan. (2020)

[23] Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., Schaus, P.: Learning optimal decision trees using constraint programming. Constraints **25**, 1–25 (2020) https://doi.org/10.1007/s10601-020-09312-3

[24] Nijssen, S., Fromont, E.: Mining optimal decision trees from itemset lattices. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '07, pp. 530–539. Association for Computing Machinery, New York, NY, USA (2007)

[25] Aglin, G., Nijssen, S., Schaus, P.: Learning optimal decision trees using caching branch-and-bound search. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 3146–3153 (2020)

[26] Aglin, G., Nijssen, S., Schaus, P.: PyDL8.5: a library for learning optimal decision trees. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pp. 5222–5224. International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan. (2020)

[27] Hu, H., Siala, M., Hebrard, E., Huguet, M.-J.: Learning optimal decision trees with maxsat and its integration in adaboost. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pp. 1170–1176. International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan (2020)

[28] Lin, J.J., Zhong, C., Hu, D., Rudin, C., Seltzer, M.I.: Generalized and scalable optimal sparse decision trees. In: ICML (2020)

[29] Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., Stuckey, P.J.: MurTree: Optimal Classification Trees via Dynamic Programming and Search (2021)

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

[31] Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4**(1), 238–252 (1962)

[32] Michini, C., Zhou, Z.: A polyhedral study of multivariate decision trees. Optimization Online (2022). https://optimization-online.org/?p=20972

[33] Vapnik, V.: Statistical Learning Theory. Wiley, Hoboken, New Jersey, U.S. (1998)

[34] Cornuéjols, G.: Combinatorial Optimization: Packing and Covering. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, United States (2001)

[35] Gleeson, J., Ryan, J.: Identifying minimally infeasible subsystems of inequalities. INFORMS J. Comput. **2**, 61–63 (1990)

[36] Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)

[37] Codato, G., Fischetti, M.: Combinatorial Benders' cuts for mixed-integer linear programming. Oper. Res. **54**(4), 756–766 (2006)

[38] Hooker, J., Ottosson, G.: Logic-based benders decomposition. Mathematical Programming **96** (2001)

[39] Brown, D.E., Pittard, C.L., Park, H.: Classification trees with optimal multivariate decision nodes. Pattern Recognition Letters **17**(7), 699–703 (1996) https:

//doi.org/10.1016/0167-8655(96)00033-5

[40] Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. CoRR **abs/cs/9408103** (1994) cs/9408103

[41] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[42] Dua, D., Graff, C.: UCI Machine Learning Repository (2017). http://archive.ics.uci.edu/ml

[43] Interpretable AI, L.: Interpretable AI Documentation (2021). https://www.interpretable.ai

[44] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023). https://www.gurobi.com