Shattering Inequalities for Learning Optimal Decision Trees

Justin J. Boutilier, Carla Michini, and Zachary Zhou

Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI, USA
{jboutilier,michini,zzhou246}@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 slow runtimes, due to weak linear programming (LP) relaxations. In this paper, 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 crucially employs binary variables expressing how each observation is routed throughout the entire tree. We then introduce a new class of valid inequalities for learning optimal multivariate decision trees. 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 leverage these valid inequalities within a Benders-like decomposition, where the master problem determines how to route each observation to a leaf node to minimize misclassification error, and the subproblem checks whether, for each branch node of the decision tree, it is possible to construct a multivariate split that realizes the given routing of observations; if not, the subproblem adds at least one of our valid inequalities to the master problem. 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.

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

1 Introduction

Decision trees are among the most popular techniques for interpretable machine learning [9]. 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 [8,23]. 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

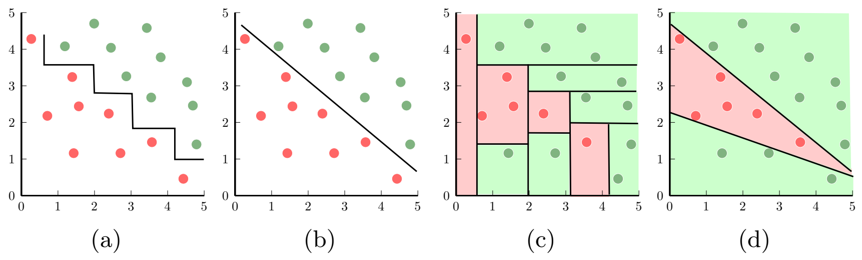


Fig. 1. (a) Eight univariate branching rules are required to correctly separate the green and red observations; (b) only one multivariate branching rule suffices. For a slightly different toy dataset: (c) and (d) show how the feature space is partitioned when using univariate and multivariate branching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

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.

Related work. The problem of learning optimal decision trees is NP-hard [22]. As a result, there exist several famous top-down induction algorithms for learning decision trees such as CART [9] and ID3 [28]. 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 [7] 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 [1,2,12,16,35,34,37]. 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 [1,16]. Moreover, MIP formulations can easily handle multivariate branching rules [7,37]. 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 [7,34,37], but this modeling technique suffers from the fact that big-M constraints notoriously lead to poor LP relaxations [36]. One notable exception is the work of Aghaei et al. [2], 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) [26,5,21,29], constraint programming (CP) [33,32], and dynamic programming [27,3,4,19,25,13] 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 [24]; 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 [6] 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 [7] and SVM1-ODT [37]). 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 $\{1, \ldots, n\}$. 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). We denote the set of branch nodes as $\mathcal{B} = \{1, \ldots, 2^D - 1\}$. Each branch node t has exactly two children nodes and 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 $\mathcal{L} = \{2^D, \ldots, 2^{D+1} - 1\}$. 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 of the decision tree $D \in \mathbb{N}$ is defined as the length of the longest path from the root note to a leaf node. The tree depth is often used 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 $\alpha \geq 0$ that penalizes the number of leaf nodes in our objective function (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 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_{kt} 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

$$
\begin{aligned}
\underset{\mathbf{c},\mathbf{d},\mathbf{w},\mathbf{z},\mathbf{a},\mathbf{b}}{\text{minimize}} \quad & \frac{1}{N}\left(1 - \sum_{i=1}^{N}\sum_{t\in\mathcal{L}} z_{it}\right) + \alpha \sum_{t\in\mathcal{B}} d_t && && \text{(1a)}\\
\text{subject to} \quad & \sum_{t\in\mathcal{L}} w_{it} = 1 && \forall i \in [N], && \text{(1b)}\\
& w_{it} = w_{i,2t} + w_{i,2t+1} && \forall i \in [N],\ t \in \mathcal{B}, && \text{(1c)}\\
& w_{i,2t+1} \le d_t && \forall i \in [N],\ t \in \mathcal{B}, && \text{(1d)}\\
& \sum_{k=1}^{K} c_{kt} = 1 && \forall t \in \mathcal{L}, && \text{(1e)}\\
& z_{it} \le w_{it} && \forall i \in [N],\ t \in \mathcal{L}, && \text{(1f)}\\
& z_{it} \le c_{y^i,t} && \forall i \in [N],\ t \in \mathcal{L}, && \text{(1g)}\\
& c_{kt} \in \{0,1\} && \forall k \in [K],\ t \in \mathcal{L}, && \text{(1h)}\\
& d_t \in \{0,1\} && \forall t \in \mathcal{B}, && \text{(1i)}\\
& w_{it} \in \{0,1\} && \forall i \in [N],\ t \in \mathcal{B} \cup \mathcal{L}, && \text{(1j)}\\
& z_{it} \in \mathbb{R} && \forall i \in [N],\ t \in \mathcal{L}, && \text{(1k)}\\
& (\mathbf{a}_t, b_t) \in \mathcal{H}_t(\mathbf{w}) && \forall t \in \mathcal{B}, && \text{(1l)}
\end{aligned}
$$

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

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

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) is derived from CART's cost-complexity measure: the misclassification rate over the training set is $1 - \frac{1}{N}\sum_{i=1}^{N}\sum_{t\in\mathcal{L}} z_{it}$; the second term weights the number of leaf nodes (to which at least one observation is directed), which is equal to the number of branch nodes (that apply a nontrivial split) plus one.

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 (1d) automatically send any incoming observations to the left child of t. Constraints (1e) assign each leaf node a class in $[K]$. Constraints (1f) and (1g) enforce the condition that if $z_{it} = 1$, then $w_{it} = 1$ and $c_{y^i,t} = 1$ (i.e., observation i is sent to leaf t and is correctly classified as y^i). Note that integrality constraints are not required for the $\mathbf{z}$ variables, since they are implied by the integrality of $\mathbf{w}$ and $\mathbf{c}$, and by the fact that at an optimal solution constraints (1f) and (1g) hold with equality. Complicating constraints (1l) 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 focus on an alternative means to characterizing the set of feasible routings $\mathbf{w}$. As mentioned in Section 1, most MIP formulations link the $\mathbf{w}$ and $(\mathbf{a}_t, b_t),\ t \in \mathcal{B}$ variables using big-M constraints. However, we have formulated the problem in such a way that these big-M constraints are not necessary. Second, we define $\mathbf{w}$ over all nodes of the decision tree, while previous literature defines $\mathbf{w}$ over only the leaf nodes [7,16,37]. 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. 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 big-M constraints, but with far fewer of them than existing formulations. For instance, OCT and OCT-H [7] require $2^D ND$ big-M constraints for defining the feasible routings $\mathbf{w}$, whereas we only require $N(2^{D+1} - 2)$. Finally, we penalize the total number of splits used as part of our objective function. 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 [7] penalizes the total number of *features* used over all splits in the tree and SVM1-ODT [37] penalizes the ℓ_1 norm of $\mathbf{a}_t$ over all splits in the tree.

3 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 $\mathcal{C}$ [31].

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):

$$\sum_{i\in I:\lambda_i=-1} w_{i,2t} + \sum_{i\in I:\lambda_i=+1} w_{i,2t+1} \leq |I|-1, \quad \forall I\in\mathcal{I},\ \lambda\in\Lambda(I),\ t\in\mathcal{B}. \tag{4}$$

The shattering inequalities (4) have the form of *packing constraints* [11] 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. First, 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 (4) associated to I is implied by an inequality (4) 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 (4) 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.

3.1 Decomposition and Separation

We decompose (1) into a master problem and an LP feasibility subproblem. The master problem is obtained from (1) by removing the complicating constraints (1l) and by (possibly) adding some valid inequalities (4). Thus, the master problem is a MIP with decision variables $\mathbf{c}, \mathbf{d}, \mathbf{w}, \mathbf{z}$. The LP feasibility subproblem includes decision variables $(\mathbf{a}_t, b_t)$, $t \in \mathcal{B}$, and verifies that, for a given assignment of $\mathbf{w}$, $\mathcal{H}_t(\mathbf{w})$ is a nonempty polyhedron for all $t \in \mathcal{B}$. Intuitively, the master problem attempts to find an optimal routing of the observations to the leaves defined by $\mathbf{w}$, and the LP feasibility subproblem attempts to find, at each branch node t, a multivariate split $(\mathbf{a}_t, b_t)$ that realizes this routing. If at some branch node there is no multivariate split that is able to route the incoming observations according to $\mathbf{w}$, then our goal is to generate an inequality (4) that is violated by $\mathbf{w}$. This inequality is added to the master problem as a feasibility cut, and the process is repeated until the subproblem becomes feasible, meaning that $\mathbf{w}$ satisfies all inequalities (4).

Next we show how to dynamically generate inequalities (4). As we shall see later, this process can be interpreted as an application of *combinatorial Benders (CB) cuts* [10,18] to the decomposition of (1) outlined above. Let $(\mathbf{c}, \mathbf{d}, \mathbf{w}, \mathbf{z})$ be a solution to the master problem. For each $t \in \mathcal{B}$, define $I(t) = \{i \in [N] : w_{it} = 1\}$ as the set of observations arriving at node t, and consider the partition of $I(t)$ into $I(2t)$ and $I(2t + 1)$. From (2) we have that $\mathcal{H}_t(\mathbf{w}) \neq \varnothing$ if and only if the observations in $I(2t)$ can be perfectly separated from the observations in $I(2t+1)$ by a multivariate split, i.e., if and only if the following system of linear inequalities in variables $(\mathbf{a}_t, b_t)$ is feasible:

$$\begin{cases} \mathbf{a}_t^\top \mathbf{x}^i + 1 \leq b_t, & \forall i \in I(2t), \\ \mathbf{a}_t^\top \mathbf{x}^i - 1 \geq b_t, & \forall i \in I(2t+1). \end{cases} \tag{5}$$

Our goal is to either certify that system (5) is feasible for all $t \in \mathcal{B}$ or, if (5) is infeasible for some $t \in \mathcal{B}$, to return an inclusion-minimal subset of observations $I' \subseteq I(t)$, such that $I' \cap I(2t)$ cannot be perfectly separated from $I' \cap I(2t+1)$ by a multivariate split. Each such subset I' corresponds to an *Irreducible Infeasible Subsystem* (IIS) of the infeasible system (5), which is defined as a subsystem of (5) that would become feasible by discarding one arbitrary inequality. Once we have found an IIS of (5) indexed by $I' \subseteq I(t)$, we add the following cut to the master problem:

$$\sum_{i \in I' \cap I(2t)} w_{i,2t} + \sum_{i \in I' \cap I(2t+1)} w_{i,2t+1} \leq |I'| - 1. \tag{6}$$

Inequality (6) is clearly violated by $\mathbf{w}$, and is a shattering inequality (4).

To find an IIS of the infeasible system (5), we construct a dual polyhedron Q_t which is obtained by applying Farkas' lemma to (5):

$$Q_t = \left\{ \mathbf{q} \in \mathbb{R}_+^{I(t)} : \quad \sum_{i \in I(2t)} q_i \mathbf{x}^i = \sum_{i \in I(2t+1)} q_i \mathbf{x}^i, \quad \sum_{i \in I(2t)} q_i = \sum_{i \in I(2t+1)} q_i = 1 \right\}. \tag{7}$$

In fact, there is a one-to-one correspondence between the IISs of (5) and the vertices of Q_t [15]. Specifically, the indices of the inequalities appearing in an IIS of (5) correspond to the support of a vertex of Q_t, and vice versa. We remark that the polyhedron Q_t has a very nice geometric interpretation. The decision variables $\mathbf{q}$ are associated with the observations indexed by $I(t) = I(2t) \cup I(2t + 1)$, and they can be interpreted as the coefficients of two convex combinations, one on the observations in $I(2t)$ and the other on the observations in $I(2t+1)$. It is evident from (7) that Q_t is nonempty if and only if there exists a point that is both in the convex hull of $\{x^i\}_{i\in I(2t)}$ and in the convex hull of $\{x^i\}_{i\in I(2t+1)}$.

Based on the above discussion, we can define a separation algorithm to dynamically generate the shattering inequalities (4). This algorithm receives as input a feasible solution $(\mathbf{c}, \mathbf{d}, \mathbf{w}, \mathbf{z})$ of the master problem, and it either establishes that $\mathbf{w}$ satisfies all inequalities (4), or it returns an inequality of family (4) that is violated by $\mathbf{w}$. Precisely, for each $t \in \mathcal{B}$, the algorithm checks the feasibility of the dual polyhedron Q_t. If Q_t is empty for all $t \in \mathcal{B}$, then by Farkas' Lemma system (5) is feasible for all $t \in \mathcal{B}$, thus the LP subproblem is feasible and we can construct an optimal solution to (1) which realizes the routing prescribed by $\mathbf{w}$. If Q_t is nonempty for some $t \in \mathcal{B}$, then by Farkas' Lemma system (5) is infeasible, and from each vertex of Q_t we can construct an IIS of (5) and a corresponding shattering inequality (6) that is violated by $\mathbf{w}$. In practice, it is possible to efficiently generate multiple inequalities (6) by finding multiple vertices of Q_t. One method for finding multiple vertices is to optimize over Q_t multiple times with different objective functions. For instance, let $\mathbf{f} \in \mathbb{Z}_+^{I(t)}$ be a counter for the number of inequalities (6) that each observation in $I(t)$ has appeared in thus far. One can repeatedly solve $\max\{\mathbf{f}^\top \mathbf{q} : \mathbf{q} \in Q_t\}$, each time updating $\mathbf{f}$ as a cut is added.

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 [30]. 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 conclude this section by observing that our formulation (1) and its decomposition can be used to generate the shattering inequalities (6) as CB cuts. CB cuts are a specialization of Hooker's logic-based Benders decomposition [18]. They are formally introduced by Codato and Fischetti [10], 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.

The shattering inequalities (6) can be interpreted as CB cuts by enforcing the complicating constraints (1l) 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 $w_{it} = 1$) 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 $\mathbf{w}$ that pertain to a specific branch node $t \in \mathcal{B}$. As a result, we can separately consider the inequality systems (5) associated with each individual branch node $t \in \mathcal{B}$.

4 Experiments

In this section, we provide two sets of numerical experiments to benchmark two implementations of our approach (S-OCT-FULL and S-OCT-BEND) with four approaches from the literature: OCT and OCT-H [7], MIP models for learning optimal univariate and multivariate trees respectively, implemented by Interpretable AI [20]; FlowOCT (solved using Benders decomposition) [2], MIP models for learning univariate trees on datasets with binary features; and DL8.5 [3], an itemset mining-based approach that uses branch-and-bound and caching.

4.1 Experimental Setup

We applied all decision tree implementations to the following fifteen commonly used datasets obtained from the UCI Machine Learning Repository [14]: (A) Balance Scale, (B) Banknote Authentication, (C) Blood Transfusion, (D) Breast Cancer, (E) Climate Model Crashes, (F) Congressional Voting Records, (G) Glass Identification, (H) Hayes-Roth, (I) Image Segmentation, (J) Ionosphere, (K) Iris, (L) Parkinsons, (M) Soybean (Small), (N) Tic-Tac-Toe Endgame, and (O) Wine. Since FlowOCT and DL8.5 require binary features, we perform an additional *bucketization* step for the numerical datasets [25]; 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. Aside from this peculiarity, we perform the standard one-hot encoding for categorical features and normalize numerical features to the $[0, 1]$ interval.

We partitioned each dataset so that 75% of the observations are used for training and 25% for testing. We tuned the complexity hyperparameter α for S-OCT and FlowOCT[1] by partitioning the training set so that two-thirds is truly used for training and one-third is used for validation. We searched over five α values: $\{0.00001, 0.0001, 0.001, 0.01, 0.1\}$. Interpretable AI automatically tunes α for OCT and OCT-H. There is no α for DL8.5.

Our experiments were programmed using Python 3.8.10 and all optimization problems were solved using Gurobi 9.5 [17] 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`.

[1] We modify the FlowOCT objective to 1) minimize error rate plus a regularization term and 2) use α rather than $\lambda \in [0, 1)$ for regularization, to be consistent with other models.

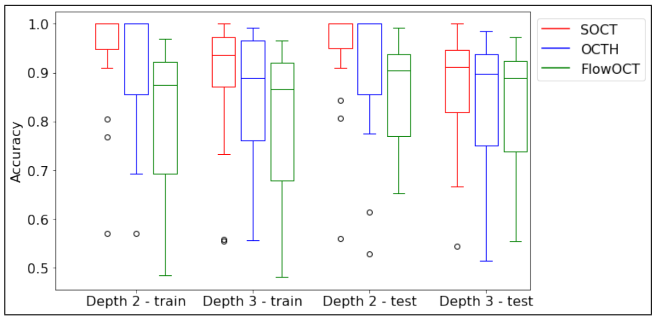
Direct MIP comparison. Our entire approach is grounded in formulation (1), which crucially uses the binary variables $\mathbf{w}$ to model how observations are routed throughout the decision tree. A straightforward way to turn our formulation (1) into a MIP formulation is to use big-M constraints to model constraints (1l) (we test the implementation without big-M constraints in the next section). We call the corresponding MIP formulation S-OCT-FULL. Our first goal is to test the strength of S-OCT-FULL against two other MIP formulations, namely OCT-H and FlowOCT. Note that OCT-H uses multivariate branching rules, while FlowOCT is tailored to datasets with binary features and uses univariate branching rules. When dealing with datasets having numerical features, we apply the preprocessing step described in Section 4.1 before applying FlowOCT.

Comprehensive Comparison. In the second set of experiments, we compare the practical performance of our approach against a wider range of other methods, both within and outside of MIP. Besides S-OCT-FULL, we also consider the decomposition approach that uses shattering inequalities described in Section 3.1, which we call S-OCT-BEND. We compare S-OCT-FULL and S-OCT-BEND against OCT and OCT-H (as implemented by Interpretable AI [20]), FlowOCT, and DL8.5. Our goal is to assess the performance of these models, even given limited training time. Oftentimes, MIP-based decision trees are able to produce near-optimal solutions within seconds, and additional solving time produces marginal improvements (if any). Therefore, for the MIP models (aside from OCT and OCT-H, as Interpretable AI's implementations of these are a black box), we terminate the solve if 60 seconds pass without a new incumbent solution being found. We also feed the MIP models a warm start; for FlowOCT, we feed CART's solution, and for S-OCT, we feed the results of a top-down greedy induction where an S-OCT stump (a tree with depth 1) is applied at each branch node so as to maximize training accuracy (similar to [7]).

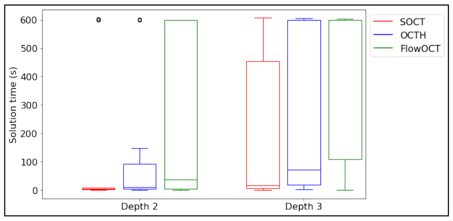
4.2 Results

Direct MIP comparison. Figure 2 displays boxplots of the training accuracy, testing accuracy, solution time, and the relative gap for S-OCT-FULL, OCT-H, and FlowOCT at depth 2 and 3. The average ± standard deviation training (testing) accuracy was 0.938±0.122 (0.870±0.136) for S-OCT-FULL, 0.912±0.146 (0.836±0.145) for OCT-H, and 0.833±0.139 (0.811±0.137) for FlowOCT. The average solution time (relative gap) was 158.7±255.1 (0.237±0.408) for S-OCT-FULL, 194.5±258.8 (0.289±0.453) for OCT-H, and 312.5±269.1 (0.463±0.366) for FlowOCT. Across all 30 instances (15 for each depth), S-OCT-FULL timed out for seven instances, OCT-H timed out for eight instances, and FlowOCT timed out for eighteen instances. Overall, we find that S-OCT-FULL achieved the highest average training and testing accuracy across all 15 datasets with the fastest average solution time and the smallest average relative gap.

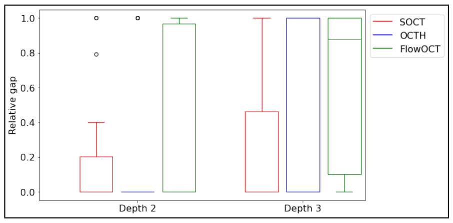
Comprehensive comparison. Table 1 provides a detailed comparison of the numerical results for each dataset and for six different model implementations.



(a)



(b)



(c)

Fig. 2. Boxplots of the training accuracy, testing accuracy, solution time, and the relative gap for S-OCT, OCT-H, and FlowOCT at depth 2 and 3.

We first compare the results between S-OCT-FULL and S-OCT-BEND to highlight the improvement in solution time offered by our benders implementation. Both models achieved similar training accuracy (0.960±0.088 for S-OCT-FULL and 0.951±0.105 for S-OCT-BEND), while S-OCT-FULL achieved a higher testing accuracy (0.901±0.115 vs. 0.876±0.145) and S-OCT-BEND achieved a faster solution time (44.6±10.7 vs. 29.2±9.9). These differences are likely due to the early stopping criteria used as part of our S-OCT-BEND implementation, which allows us to achieve similar testing performance with shorter solution times as compared to S-OCT-FULL.

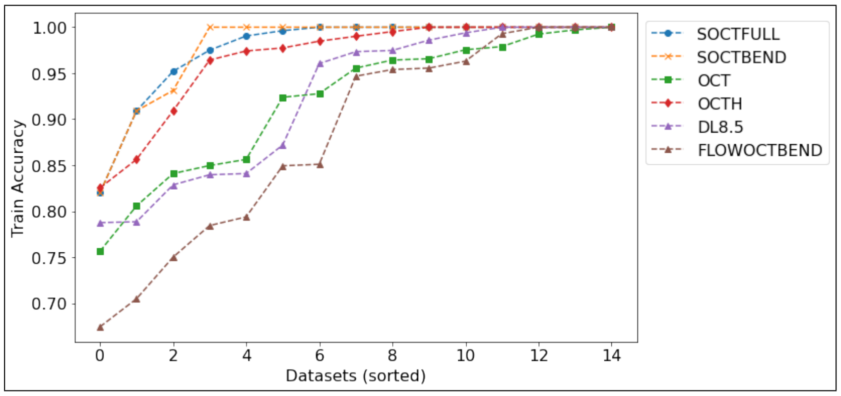
Next, we compare the in-sample accuracy achieved by all models. We find that S-OCT-FULL performed best (0.960±0.088), followed by S-OCT-BEND (0.951±0.105) and OCT-H (0.948±0.086). The remaining models all performed significantly worse: OCT (0.882±0.118), DL8.5 (0.883±0.121), and FlowOCT (0.859±0.140). We observe that the multivariate models (S-OCT-FULL, S-OCT-BEND, and OCT-H) significantly outperform the univariate models (DL8.5, FlowOCT, OCT). Intuitively, this makes sense because univariate models require more depth to achieve comparably complex branching rules (see Figure 1).

We find similar results for out-of-sample accuracy, where S-OCT-FULL performed best (0.901±0.115), followed closely by OCT-H (0.885±0.115) and S-OCT-BEND (0.876±0.145). The remaining models achieve an average testing accuracy of 0.849±0.125 (OCT), 0.846±0.126 (DL8.5), and 0.821±0.155 (FlowOCT). Finally, we compare solutions times between all approaches. The average solution time was 44.6±56.3 for S-OCT-FULL, 29.2±30.0 for S-OCT-BEND, 0.7±1.7 for OCT, 15.0±18.6 for OCT-H, 18.0±46.1 for DL8.5, and 57.4±38.6 for FlowOCT. Although our models had the second and third slowest solutions times, they are able to find the (provably) optimal solution in 9/15 instances.

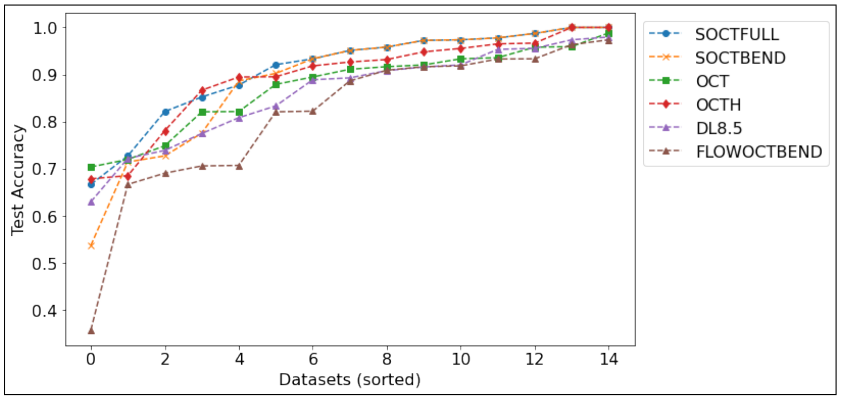
Figure 3 displays line plots of the training accuracy, testing accuracy, and solution time across all 15 datasets (sorted for clarity) and for all six models at depth four. The line plots visualize the results in Table 1 for depth four; we see that both S-OCT-FULL and S-OCT-BEND achieve the highest training accuracy for all datasets and the highest testing accuracy for 12/15 datasets. Our models are comparable in terms of solution time for 9/15 datsets; for the remaining seven datasets, our models are slightly slower.

5 Conclusion

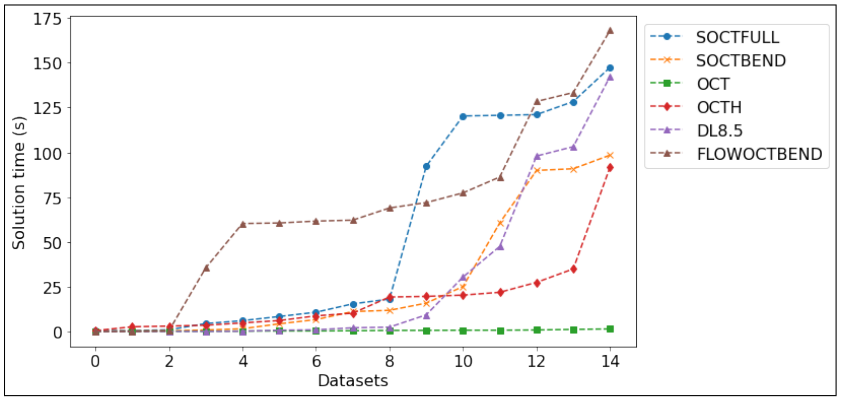
We proposed a new MIP formulation for the optimal decision tree problem. Our approach directly deals with numerical features and leverages the higher modeling power of multivariate branching rules. We also introduced a new class of valid inequalities and an exact decomposition approach that uses these 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 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. Finally, we note that our formulation and the shattering inequalities (4) 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. 3. Line plots across all 15 datasets (sorted for clarity) of the training accuracy, testing accuracy, and solution time for all six models at depth four.

Table 1. Detailed summary of comprehensive comparisons.

	Dataset														
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)	(J)	(K)	(L)	(M)	(N)	(O)
Observations (N)	468	1029	561	426	405	326	160	132	210	263	112	146	35	718	133
Features (p)	20	4	4	30	18	48	9	15	19	34	4	22	72	27	13
Buckets	N/A	1303	105	3499	1149	N/A	487	N/A	1657	1779	35	892	N/A	N/A	456
Classes (K)	3	2	2	2	2	2	6	3	7	2	3	2	4	2	3
Results for $D = 2$															
	In-sample accuracy (%)														
SOCTFULL	100.0	100.0	80.57	100.0	99.26	100.0	70.0	90.91	57.14	100.0	100.0	100.0	100.0	100.0	100.0
SOCTBEND	100.0	100.0	80.57	100.0	100.0	100.0	74.38	90.91	57.14	100.0	100.0	97.26	100.0	100.0	100.0
OCT	68.59	93.0	79.5	96.01	93.33	95.4	69.38	60.61	57.14	90.87	96.43	91.1	100.0	65.74	93.98
OCTH	99.79	99.51	82.17	97.18	98.02	100.0	73.12	90.15	57.14	95.82	96.43	97.26	97.14	100.0	96.99
DL8.5	68.59	93.0	79.68	96.01	93.83	96.01	69.38	60.61	57.14	91.25	96.43	91.1	100.0	72.14	96.99
FLOWOCT	68.59	91.55	78.07	94.13	91.11	95.4	64.38	60.61	42.86	90.87	96.43	91.1	100.0	72.14	96.99
	Out-of-sample accuracy (%)														
SOCTFULL	98.73	100.0	73.26	95.1	95.56	97.25	59.26	78.57	56.14	90.91	97.37	79.59	100.0	95.83	97.78
SOCTBEND	98.73	100.0	73.26	95.1	93.33	97.25	59.26	82.14	55.57	88.64	97.37	77.55	100.0	95.83	97.78
OCT	66.88	91.84	70.59	94.41	94.07	96.33	55.56	67.86	56.57	92.05	89.47	89.8	100.0	64.17	84.44
OCTH	98.09	98.54	75.94	96.5	94.81	89.91	53.7	78.57	56.57	92.05	89.47	93.88	91.67	95.42	95.56
DL8.5	66.88	91.55	69.52	94.41	94.07	95.41	55.56	75.0	56.52	90.91	89.47	93.88	100.0	65.83	95.56
FLOWOCT	66.88	91.55	70.59	93.01	92.59	96.33	55.56	67.86	42.71	92.05	89.47	91.84	100.0	65.83	95.56
	Computational time (s)														
SOCTFULL	1.27	6.36	108.15	4.55	55.49	0.81	149.88	8.68	1.03	109.04	0.2	33.28	0.21	10.69	0.28
SOCTBEND	0.64	5.48	106.57	3.71	11.39	0.43	98.43	20.65	19.34	30.55	0.14	137.43	0.16	9.69	0.18
OCT	8.48	0.76	0.16	0.54	0.29	0.3	0.09	0.07	0.19	0.33	0.03	0.17	7.83	0.33	0.09
OCTH	25.0	10.3	3.01	15.81	10.9	19.06	2.72	2.39	4.38	14.69	0.39	4.64	3.62	50.55	2.49
DL8.5	0.0	4.5	0.03	35.93	3.56	0.01	0.59	0.0	8.27	6.94	0.0	1.64	0.0	0.01	0.33
FLOWOCT	1.07	65.23	60.13	65.97	64.45	2.75	60.41	0.47	60.72	72.15	0.23	93.48	0.51	61.5	61.9
Results for $D = 3$															
	In-sample accuracy (%)														
SOCTFULL	100.0	100.0	81.46	100.0	100.0	100.0	90.62	90.91	98.57	100.0	100.0	95.21	100.0	100.0	100.0
SOCTBEND	100.0	100.0	81.46	100.0	99.26	100.0	85.0	90.91	57.14	100.0	100.0	100.0	100.0	100.0	100.0
OCT	73.08	97.76	80.57	98.36	95.56	95.4	76.25	72.73	84.76	93.54	98.21	98.63	100.0	75.63	99.25
OCTH	100.0	100.0	82.53	99.3	99.01	100.0	83.12	89.39	93.33	98.1	96.43	100.0	100.0	99.44	97.74
DL8.5	74.15	93.68	81.64	96.48	93.58	97.85	80.0	74.24	65.24	92.02	100.0	100.0	100.0	78.69	100.0
FLOWOCT	73.72	93.97	80.75	95.54	96.3	97.24	74.38	74.24	57.14	92.4	100.0	95.89	100.0	76.6	99.25
	Out-of-sample accuracy (%)														
SOCTFULL	98.73	100.0	73.26	95.1	93.33	97.25	70.37	85.71	93.0	89.77	97.37	87.76	100.0	95.83	97.78
SOCTBEND	98.73	100.0	73.26	95.1	95.56	97.25	53.7	71.43	56.0	88.64	97.37	81.63	100.0	95.83	97.78
OCT	68.79	96.5	74.87	95.8	93.33	96.33	64.81	75.0	81.43	92.05	97.37	97.96	91.67	72.92	97.78
OCTH	95.54	100.0	77.54	95.1	94.07	92.66	61.11	82.14	83.57	96.59	89.47	97.96	91.67	96.25	86.67
DL8.5	71.97	92.42	72.73	90.91	90.37	93.58	68.52	89.29	63.71	89.77	97.37	95.92	100.0	73.75	93.33
FLOWOCT	71.97	93.0	75.4	91.61	91.85	93.58	62.96	89.29	56.67	92.05	97.37	93.88	91.67	71.67	91.11
	Computational time (s)														
SOCTFULL	2.28	6.96	97.09	6.3	13.52	1.68	182.6	51.54	148.03	88.04	0.43	109.17	0.32	12.57	0.48
SOCTBEND	0.98	5.87	97.63	3.89	82.26	0.56	89.06	60.94	60.54	20.63	0.2	20.55	0.17	9.91	0.27
OCT	0.5	0.65	0.19	1.0	0.51	0.46	0.16	0.1	0.32	0.73	0.04	0.26	0.05	0.69	0.14
OCTH	21.18	10.97	4.07	23.19	16.58	19.12	4.37	3.03	10.66	24.26	0.49	7.56	2.97	77.92	2.99
DL8.5	0.02	0.01	2.69	2.19	1.35	0.13	244.93	0.01	0.53	2.67	0.01	54.99	0.0	0.06	3.07
FLOWOCT	77.7	88.52	60.61	109.1	62.81	63.15	61.36	20.59	75.47	66.38	4.08	87.22	0.01	60.79	60.86
Results for $D = 4$															
	In-sample accuracy (%)														
SOCTFULL	100.0	100.0	82.0	100.0	100.0	100.0	97.5	90.91	99.05	99.62	100.0	95.21	100.0	100.0	100.0
SOCTBEND	100.0	100.0	82.0	100.0	100.0	100.0	93.12	90.91	100.0	100.0	100.0	100.0	100.0	100.0	100.0
OCT	75.64	99.71	80.57	97.89	95.56	97.55	85.62	84.09	92.38	92.78	96.43	96.58	100.0	84.96	99.25
OCTH	100.0	100.0	82.53	97.42	99.01	100.0	85.62	90.91	99.52	98.48	96.43	100.0	100.0	100.0	97.74
DL8.5	78.85	97.47	83.96	98.59	96.05	99.39	78.75	84.09	82.86	97.34	100.0	100.0	100.0	87.19	100.0
FLOWOCT	75.0	96.31	78.43	99.3	95.56	95.4	79.38	67.42	70.48	94.68	100.0	84.93	100.0	85.1	100.0
	Out-of-sample accuracy (%)														
SOCTFULL	98.73	100.0	72.73	95.1	93.33	97.25	66.67	82.14	92.1	85.23	97.37	87.76	100.0	95.83	97.78
SOCTBEND	98.73	100.0	72.73	95.1	93.33	97.25	53.7	71.43	90.24	88.64	97.37	77.55	100.0	95.83	97.78
OCT	71.97	98.83	74.87	95.8	93.33	93.58	70.37	82.14	87.9	92.05	89.47	95.92	91.67	82.08	91.11
OCTH	95.54	100.0	78.07	96.5	94.81	92.66	68.52	67.86	89.52	93.18	89.47	91.84	100.0	96.67	86.67
DL8.5	73.89	95.34	72.19	91.61	88.89	90.83	62.96	89.29	77.48	92.05	97.37	97.96	83.33	80.83	95.56
FLOWOCT	70.7	93.29	70.59	90.91	93.33	96.33	66.67	35.71	69.05	88.64	97.37	91.84	91.67	82.08	82.22
	Computational time (s)														
SOCTFULL	6.05	10.74	92.25	8.37	15.42	4.43	128.31	120.29	147.42	120.64	0.54	121.07	0.55	18.08	0.94
SOCTBEND	1.53	6.63	90.82	4.29	11.81	0.83	89.9	60.61	98.44	15.82	0.32	25.06	0.19	11.12	0.36
OCT	0.72	0.85	0.29	1.17	0.6	0.63	0.22	0.15	0.42	0.71	0.05	0.31	0.06	1.44	0.18
OCTH	27.4	10.22	4.79	20.29	21.87	19.23	6.11	3.49	19.57	34.92	0.49	8.72	2.63	91.83	3.02
DL8.5	0.15	0.05	142.38	97.92	30.4	2.07	0.96	0.03	9.28	103.14	0.01	47.48	0.0	0.72	2.44
FLOWOCT	128.38	86.22	168.05	77.36	61.6	60.54	71.92	133.35	62.13	68.93	0.03	35.45	0.01	60.25	0.37

References

1. Aghaei, S., Azizi, M.J., Vayanos, P.: Learning optimal and fair decision trees for non-discriminative decision-making (2019)
2. Aghaei, S., Gomez, A., Vayanos, P.: Learning optimal classification trees: Strong max-flow formulations (2020)
3. 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 (Apr 2020)
4. 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 (7 2020), demos
5. Avellaneda, F.: Efficient inference of optimal decision trees. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 3195–3202 (Apr 2020)
6. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numer. Math. **4**(1), 238–252 (Dec 1962)
7. Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning **106** (07 2017)
8. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
9. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and regression trees. CRC press (1984)
10. Codato, G., Fischetti, M.: Combinatorial benders' cuts for mixed-integer linear programming. Oper. Res. **54**(4), 756–766 (2006)
11. Cornuéjols, G.: Combinatorial Optimization: Packing and Covering. CBMS-NSF Regional Conference Series in Applied Mathematics (2001)
12. Dash, S., Günlük, O., Wei, D.: Boolean decision rules via column generation (2020)
13. 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)
14. Dua, D., Graff, C.: UCI machine learning repository (2017), `http://archive.ics.uci.edu/ml`
15. Gleeson, J., Ryan, J.: Identifying minimally infeasible subsystems of inequalities. INFORMS J. Comput. **2**, 61–63 (1990)
16. Gunluk, O., Kalagnanam, J., Li, M., Menickelly, M., Scheinberg, K.: Optimal generalized decision trees via integer programming (2019)
17. Gurobi Optimization, L.: Gurobi optimizer reference manual (2021), `http://www.gurobi.com`
18. Hooker, J., Ottosson, G.: Logic-based benders decomposition. Math. Prog. **96** (03 2001)
19. 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 (7 2020)
20. Interpretable AI, L.: Interpretable ai documentation (2021), `https://www.interpretable.ai`
21. 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 International Publishing, Cham (2020)

22. Laurent, H., Rivest, R.L.: Constructing optimal binary decision trees is np-complete. Information processing letters **5**(1), 15–17 (1976)
23. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R news **2**(3), 18–22 (2002)
24. 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. PMLR (2020)
25. Lin, J.J., Zhong, C., Hu, D., Rudin, C., Seltzer, M.I.: Generalized and scalable optimal sparse decision trees. In: ICML (2020)
26. Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J.: Learning optimal decision trees with sat. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. pp. 1362–1368. International Joint Conferences on Artificial Intelligence Organization (7 2018)
27. 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. p. 530–539. KDD '07, Association for Computing Machinery, New York, NY, USA (2007)
28. Quinlan, J.R.: Induction of decision trees. Machine learning **1**(1), 81–106 (1986)
29. 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 (May 2021)
30. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)
31. Vapnik, V.: Statistical learning theory. Wiley (1998)
32. Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.G., Schaus, P.: Learning optimal decision trees using constraint programming. Constraints **25**, 1–25 (12 2020). https://doi.org/10.1007/s10601-020-09312-3
33. 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 (7 2020)
34. 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
35. 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 International Publishing, Cham (2017)
36. Wolsey, L.: Integer Programming. Wiley Series in Discrete Mathematics and Optimization, Wiley (1998)
37. 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)