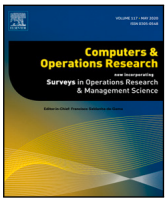
Computers & Operations Research 166 (2024) 106629

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Optimal classification trees with leaf-branch and binary constraints

Enhao Liu [a,*], Tengmu Hu [b], Theodore T. Allen [b], Christoph Hermes [c]

[a] University of Maryland Medical System, 920 Elkridge Landing Rd, Linthicum Heights, 21090, MD, United States
[b] The Ohio State University, 1971 Neil Avenue – 210 Baker Systems, Columbus, 43221, OH, United States
[c] Rosen Group, Am Seitenkanal 8, Lingen(EMS), 49811, Lower Saxony, Germany

ARTICLE INFO

Keywords:
Decision tree model
Machine learning
Mixed integer optimization

ABSTRACT

Using empirical models to predict whether sections within pipes have defects can save inspection costs and, potentially, avoid oil spills. Optimal Classification Tree (OCT) formulations offer potentially desirable combinations of interpretability and prediction accuracy on unseen pipes. Approaches based on powerful state-of-the-art OCT formulations have enabled researchers to solve decision tree problems optimally instead of using traditional sub-optimal greedy approaches. Yet, the recently proposed formulations also have limitations. Some of the most recent formulations require a large number of decision variables and constraints leading to computational inefficiencies. Previous formulations have optimal solutions with undesirable or invalid tree structures which may depend on the particular software implementation. Additionally, some formulations always grow a full tree even when desirable parsimonious tree options are available. This article proposes the Modified Optimal Classification Tree (M-OCT) formulation with novel leaf-branch-interaction constraints, which could stabilize the previous formulation and reduce the chance of invalid tree structures when generating optimal trees. By incorporating the idea of binary encoding of thresholds from a previous article, we reduce the total number of binary variables. We then extend M-OCT to construct a novel formulation called Binary Node Penalty Optimal Classification Tree (BNP-OCT) with binary splits and node complexity constraints, which support efficiency in standard branch-and-cut solvers and prevents the overfitting issue when learning the optimal tree models. We compare the proposed methods with alternatives including standard formulations using 15 standard data sets. In addition, we use 750 test cases to compare the computational stability of pre-existing formulations to those involving the proposed leaf-branch constraints. We demonstrate that the proposed formulation offers advantages in accuracy, computational efficiency, and structural stability. We also describe how the proposed methods are able to achieve 94% classification accuracy on balanced test sets for unseen pipes.

1. Introduction

Decision tree models are one of the most important and popular approaches in the field of machine learning because of their versatility in solving classification and regression problems and their built-in interpretability. These single tree models compare favorably with ensemble learning and deep learning models (Bertsimas and Shioda, 2007; Bertsimas and Dunn, 2017). Our motivating application here is the classification of sections of pipelines relating to dangerous corrosion. There are over seven million kilometers of oil and gas pipelines in various states of degradation. The problems that we study here involve magnetic flux leakage data such that the estimation problems can be viewed as a type of image analysis. The associated classifications are typically only trusted to human inspectors, but deep learning empirical modeling methods have been studied for this application (Yang et al., 2019). In our experience, only a highly accurate and interpretable model is likely to be accepted by real-world inspectors in the short term; i.e., deep learning models might not be acceptable.

A decision tree model is represented by a binary branching structure with internal nodes (a.k.a., branch nodes) and leaf nodes, and it is generated by recursively partitioning the feature space of the training data set at branch nodes and assigning prediction labels at leaf nodes. Constructing an optimal decision tree is proven to be an NP-hard problem (Laurent and Rivest, 1976) with the possibility to express the formulation in multiple ways (Carrizosa and Romero Morales, 2013). Therefore, the widely used approaches are heuristics such as greedy methods including the popular Classification and Regression Trees (CART) (Breiman et al., 1984), ID3 (Quinlan, 1986), and C4.5 (Quinlan, 1993). In recent years, there have been genetic programming approaches to fitting trees near optimally (Santos

* Corresponding author.
E-mail addresses: enhao.liu@umm.edu (E. Liu), hu.713@osu.edu (T. Hu), allen.515@osu.edu (T.T. Allen), chermes@rosen-group.com (C. Hermes).

https://doi.org/10.1016/j.cor.2024.106629
Received 6 February 2023; Received in revised form 18 March 2024; Accepted 19 March 2024
Available online 20 March 2024
0305-0548/© 2024 Elsevier Ltd. All rights reserved.

et al., 2022) and heuristics based on column generation (Firat et al., 2020). There is also increasing attention on learning the optimal decision tree based on various mathematical optimization models. Related formulations include linear programming models with Extreme Point Tabu Search algorithms (Bennett and Blue, 1996), Mixed-Integer Optimization (MIO), continuous non-linear formulations (Blanquero et al., 2021), and quadratic optimization (Bertsimas and Shioda, 2007), constraint programming and linear programming to minimize the tree size (Bessiere et al., 2009), constraint programming with the use of caching (Verhaeghe et al., 2020), "itemset" mining techniques to generate optimal trees (Aglin et al., 2020), and most recently, various Mixed-Integer Linear Programming (MILP) techniques for finding optimal decision trees (Bertsimas and Dunn, 2017; Bertsimas et al., 2019; Verwer and Zhang, 2017, 2019; Aghaei et al., 2019, 2020; Günlük et al., 2021). Here, we also propose a novel and complete formulation of Optimal Classification Tree models seeking to achieve improved accuracy and efficiency trade-offs. Apparently, some of the previous MIP formulations were incomplete. This incompleteness was likely unnoticed because the problems were solved with warm starts and heuristics.

Many recent tree modeling approaches use powerful state-of-the-art MIO solvers such as Gurobi (Gurobi Optimization, LLC, 2020) and CPLEX (ILOG CPLEX, IBM, 2020). The related MILP-based approaches are able to obtain better decision trees compared with the greedy approaches according to the numerical results. Specifically, (Bertsimas and Dunn, 2017) proposed a generic MILP model to learn the Optimal Classification Tree (OCT) by minimizing the misclassification counts and the number of nodes to split. Bertsimas et al. (2019) extended their OCT models to develop optimal prescriptive trees and proposed a heuristic approach called "coordinate descent" to repeatedly optimize the splits in the tree. Based on OCT models, Günlük et al. (2021) developed a MILP-based formulation for optimal decision trees with a pre-specified size and a special structure for categorical features in data sets. Verwer and Zhang (2017) incorporated flexible constraints and objectives in their proposed MILP model to learn a discrimination-aware classification tree. Aghaei et al. (2019) proposed MIP-based formulations for learning optimal and fair decision trees, which are designed for non-discriminative decision-making. Recently, Aghaei et al. (2020) developed a max-flow-based MIP formulation with Bender's decomposition for learning the optimal decision trees when feature values are binary. As stated in Verwer and Zhang (2019), the main limitation of these MILP-based models is that the size of constraints and binary variables depends on the number of training data points, which would result in computational issues when the data size is large. Hence, Verwer and Zhang (2019) developed a new formulation with a binary encoding program called BinOCT. They utilized a binary search procedure encoded for selecting split thresholds with a type of big-M constraint, which only requires a small number of binary variables, and the total number of binary variables is independent of the number of data points. However, the proposed BinOCT model (Verwer and Zhang, 2019) does not take the model complexity into account. The optimal tree learned from BinOCT would always be a full tree, which leads to potentially an overfitting issue and less interpretability.

In this article, we propose two types of novel constraints. First, we propose novel node-complexity-binary-split constraints to extend the BinOCT formulation (Verwer and Zhang, 2019) to include node complexity. The work presented by Verwer and Zhang (2019) introduces a novel approach to optimize feature selection by organizing feature values with identical labels into bins. These bins are determined based on feature thresholds, where each threshold delineates the range of values assigned to a specific bin. Consequently, each bin plays a crucial role in determining the direction of classification of data points. To operationalize this concept, a binary split method is employed, involving the iterative division of bins into left and right directions. Upon the segmentation of bins into left and right sides, the direction of data points is effectively established. However, a critical aspect that remains unaddressed in Verwer and Zhang (2019) study is the issue of node complexity. The conventional format involves binary splits occurring at each branch node of the tree. Ideally, this constant binary split at every branch node raises concerns related to potential overfitting. To mitigate this, an additional variable is introduced to determine whether a node should undergo a binary split. This introduces a set of node-complexity-binary-split constraints aimed at preventing binary splits at every branch node, thus addressing potential overfitting issues. Section 2.1 provides a detailed example and explanation of this modification. Second, we propose leaf-branch-interaction constraints to address potentially spurious objective function values in previous OCT formulation (Bertsimas and Dunn, 2017). In addressing the challenge of node complexity, Bertsimas and Dunn (2017) manage this issue by assigning all data points to the rightmost leaf nodes. Despite this approach, the experimental investigation of the proposed formulation reveals that data points continue to be classified, even when branch nodes are not intended to perform this classification. To rectify this occurrence, we introduce an additional set of constraints designed to prevent such situations. These constraints, referred to as leaf-branch-interaction constraints, are implemented to ensure that the undesired classification of data points by branch nodes is effectively mitigated. Consequently, these two types of novel constraints lead to two new MILP formulations for training Optimal Classification Trees: (1) Binary Node Penalty Optimal Classification Trees (BNP-OCT) with new node-complexity-binary-split and leaf-branch-interaction constraints, which is the extension of the M-OCT with the binary encoding methods from Verwer and Zhang (2019) and (2) Modified Optimal Classification Trees (M-OCT) with new leaf-branch-interaction constraints to ensure tree-valid solutions, which is a variant of the original OCT model (Bertsimas and Dunn, 2017). Our proposed BNP-OCT model not only requires a relatively small number of binary decision variables but also takes model complexity into account, which could prevent the overfitting issue when learning the optimal tree models.

The remainder of this article is organized as follows. In Section 2, we present the key elements of learning the Optimal Classification Trees along with the novel and generic BNP-OCT formulation. In Section 3, results are presented to show that the original OCT formulation does not always produce valid tree solutions, and new leaf-branch-interaction constraints are proposed that stabilize the original formulation. In Section 4, we conduct experiments on multiple benchmark data sets applied to the proposed M-OCT and BNP-OCT models and compare results with the prominent alternative approach. In Section 5, the advantages of the proposed models, limitations, and opportunities for future research are described. The appendix describes a modified version of the original formulation with the leaf-branch constraints. It might be computationally relevant for specific situations.

2. Problem definition

In the classification tree modeling problem, the objective is to construct a binary tree of maximum depth $D_{\max}$ to minimize the model misclassification errors and the model complexity given a training data set with N data points, F features, and C possible class labels. We denote (x_{ij}, y_i) as each data point in the training data set, where $x_{ij} \in \mathbb{R}$ refers to the feature j's value of data point i and $y_i \in K = \{1, \dots, C\}$ refers to a class label, $\forall i \in I = \{1, \dots, N\}$, $\forall j \in J = \{1, \dots, F\}$. For a binary tree of maximum depth $D_{\max}$, the nodes in the tree are divided into two sets: branch nodes $T_B = \{1, \dots, \lfloor T/2 \rfloor\}$ and leaf nodes $T_L = \{\lfloor T/2 \rfloor + 1, \dots, T\}$, where $T = 2^{D_{\max}+1} - 1$ represents the total number of nodes. Fig. 1 shows a full decision tree with a maximum depth of 2 and the branch nodes in this tree are $\{1, 2, 3\}$ and leaf nodes are $\{4, 5, 6, 7\}$. As discussed before, the OCT model proposed by Bertsimas and Dunn (2017) provides a generic formulation for the Optimal Classification Tree problem. It mainly aims to find the "split rules" at each branch node (i.e., which feature to split on, and what the threshold value is), and to assign prediction labels to leaf nodes such

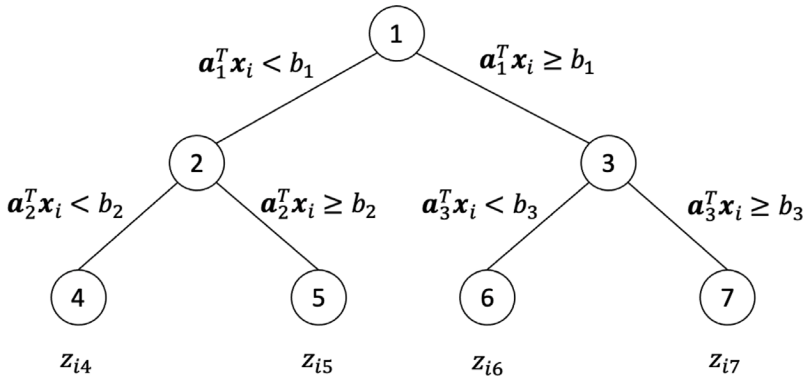


Fig. 1. A full decision tree with maximum depth $D_{\max} = 2$.

that the total count of misclassification errors and the total number of nodes for branching are minimized. The "split rules" are characterized by the following variables:

- a_{ji}: indicating which feature $j \in J$ is selected to split at branch node $t \in T_B$.
- b_t: determining the threshold value for the feature to split at branch node $t \in T_B$.

As shown in Fig. 1, a data point i with feature values $\boldsymbol{x}_i \in \mathbb{R}^F$ traverses from the root node to the leaf node by following left-direction path if $\boldsymbol{a}_t^T \boldsymbol{x}_i < b_t$, otherwise following right-direction path. The binary variable z_{it} is indicating if the data point i reaches to the leaf node t. Note that we only consider the univariate decision tree problems, i.e., only a single feature is allowed to split at each branch node. When all data points go through the tree, we need to assign a class label c_t to each leaf node t. Clearly, the class label c_t should be the most common label among data points that reach the leaf node t. Therefore, we use a similar notation from Bertsimas and Dunn (2017) to formulate the first MILP model called Modified-OCT (M-OCT) by considering new leaf-branch-interaction constraints to ensure a valid structure of a tree. To preview, in Section 3, we show that the original OCT formulation (Bertsimas and Dunn, 2017) does not always produce a valid tree solution. As an example of an invalid solution, a branch node is turned off, but data points are still assigned to all the leaf nodes. The detailed definitions of variables and the formulation of our proposed M-OCT are presented in Section 2.2 and Appendix.

According to Bertsimas et al. (2019) and Verwer and Zhang (2019), a major challenge for the original OCT and new M-OCT formulations is that the total number of binary decision variables depends to a large extent on the number of data points in the training data set. The majority $O(N2^{D_{\max}})$ of binary variables is attributed to assignment variables z_{it}, i.e., data point $i \in I$ is assigned to leaf node $t \in T_L$. Therefore, Verwer and Zhang (2019) proposed the BinOCT formulation that leverages the binary encoding approach to define the threshold variables in the branch nodes, which is able to significantly reduce the total number of binary variables. Their approach is motivated by Boolean decision thresholds. Assume that a tree with a depth of 1 is trained and there is only one binary threshold variable at branch node 1, i.e., b_1. Therefore, we will have $z_{i2} + z_{i3} = 1$, $z_{i2} + b_1 \leq 1$, and $z_{i3} - b_1 \leq 0$. That is, data point i will reach leaf node 2 (left direction) if $b_1 = 0$, and data point i will reach leaf node 3 (right direction) if $b_1 = 1$.

Note that the assignment variables z_{i2} and z_{i3} are forced to be 0 or 1 according to those constraints, and essentially, they are controlled by the binary decisions in b_1. Instead of modeling the threshold variables b_t as continuous variables, Verwer and Zhang (2019) define b_{tr} as binary encoding threshold variables by utilizing the fact that there are only a limited number of thresholds for each feature to split. Although this binary approach can often model the tree problem using relatively few binary variables, the original set of constraints associated with

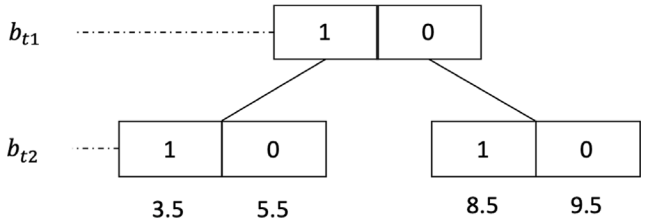


Fig. 2. The binary encoding threshold variables.

the binary variables in BinOCT do not take the node complexity into account. This omission always generates a full tree and potentially leads to an overfitting issue. Therefore, we formulate a new MILP model called BinNodePenalty-OCT (BNP-OCT) that extends the M-OCT with binary splits, leaf-branch-interaction, and node complexity constraints. We illustrate the key ideas of BNP-OCT in Section 2.1 and explicitly present the notation and formulations in Sections 2.2 and 2.3.

2.1. Node-complexity-binary-split constraints illustration

We now describe the original binary split constraints proposed by Verwer and Zhang (2017) and then present our novel node-complexity-binary-split constraints. These are important in our second proposed formulation and permit greatly improved computational efficiencies in many relevant cases. Assume that we have a simple example with $N = 10$ data points, $F = 1$ feature, $C = 2$ possible class labels, i.e., $x_{ij} = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ (note that there is only one feature, i.e., $j = 1$), and $y_i = 1, 1, 1, 0, 0, 1, 1, 1, 0, 1$ for $i \in \{1, 2, \ldots, 10\}$.

We aim to find a classification tree of maximum depth $D_{\max} = 1$. Since there is only one feature and one depth of the tree, we only need to know what the threshold value is in the branch node 1 and how to assign data points to the leaf nodes 2 and 3 to minimize the misclassification counts. Following Verwer and Zhang (2019), we define the possible thresholds of feature j as the mean value of two consecutive pre-sorted data points x_{ij} with different class labels. Hence, there are four possible thresholds {3.5, 5.5, 8.5, 9.5} in this example. We use $\log_2 4 = 2$ binary encoding threshold variables b_{t1} and b_{t2} to select the possible threshold value at branch node $t = 1$ as shown in Fig. 2.

Firstly, we could observe that data points $x_{ij} \geq \max(threshold)$ would never reach leaf node 2 (left direction), which leads to the constraint: $\sum_i z_{i2} \leq 0$, for $i \in \{10\}$. Similarly, data points $x_{ij} \leq \min(threshold)$ would never reach leaf node 3 (right direction), which leads to the constraint: $\sum_i z_{i3} \leq 0$, for $i \in \{1, 2, 3\}$.

When $b_{t1} = 1$, the threshold could be either 3.5 or 5.5 such that data points $5.5 \leq x_{ij} \leq \max(threshold)$ cannot reach leaf node 2 (left direction), which results in the constraints: $z_{i2} + b_{t1} \leq 1$, for $i \in$

$\{6,7,8,9\}, t \in \{1\}$. We could also sum these constraints for each data point: $\sum_i z_{i2} + 4b_{t1} \leq 4$, for $i \in \{6,7,8,9\}, t \in \{1\}$. Similarly, when $b_{t1} = 0$, the threshold could be either 8.5 or 9.5 such that data points $\min(threshold) \leq x_{ij} \leq 8.5$ cannot reach leaf node 3 (right direction), which results in the sum constraints: $\sum_i z_{i3} - 5b_{t1} \leq 0$, for $i \in \{4,5,6,7,8\}, t \in \{1\}$.

Depending on b_{t1}, the variable b_{t2} is used to control the left or right direction for the remaining data points in the ranges of $[3.5, 5.5]$ and $[8.5, 9.5]$. When $b_{t2} = 1$, the threshold could be either 3.5 or 8.5 such that data points $8.5 \leq x_{ij} \leq \max(threshold)$ cannot reach leaf node 2 (left direction), which results in the sum constraint: $\sum_i z_{i2} + b_{t2} \leq 1$, for $i \in \{9\}, t \in \{1\}$. Note that this constraint is independent of the variable b_{t1}. For the remaining data points $3.5 \leq x_{ij} \leq 5.5$, we will have $\sum_i z_{i2} + 2(b_{t1} + b_{t2}) \leq 4$, for $i \in \{4,5\}, t \in \{1\}$. That is, when $b_{t1} = 1$ and $b_{t2} = 1$, data points $3.5 \leq x_{ij} \leq 5.5$ would never follow the left direction. Similarly, when $b_{t2} = 0$, the threshold could be either 5.5 or 9.5 such that data points $\min(threshold) \leq x_{ij} \leq 5.5$ cannot reach leaf node 3 (right direction), which results in the sum constraint: $\sum_i z_{i3} - 2b_{t2} \leq 0$, for $i \in \{4,5\}, t \in \{1\}$. For the remaining data points $8.5 \leq x_{ij} \leq 9.5$, we will have $\sum_i z_{i3} - b_{t1} - b_{t2} \leq 0$, for $i \in \{9\}, t \in \{1\}$.

In our BNP-OCT formulation, based on Verwer and Zhang (2019), we extend these split constraints to consider arbitrary ranges of threshold values to generate $bin(j)$ range sets for each feature j. In addition, taking the node complexity into account, we incorporate the binary branching variable d into constraints. The generalized and novel node-complexity-binary-split constraints are as follows. The first two constraints relate to the binary split variables directly for both left and right paths respectively:

$$\sum_{i \in I_L(j,q)} z_{it} + M a_{jm} + M \sum_{r \in B_L(j,q)} b_{mr} \leq \left(M + \sum_{r \in B_L(j,q)} M \right) \cdot d_m, \quad \forall t \in T_L, \forall m \in A_L(t), \forall j \in J, \forall q \in bin(j) \tag{1}$$

$$\sum_{i \in I_R(j,q)} z_{it} + M' a_{jm} - M' \sum_{r \in B_R(j,q)} b_{mr} \leq M', \quad \forall t \in T_L, \forall m \in A_R(t), \forall j \in J, \forall q \in bin(j) \tag{2}$$

where M refers to the number of data points in the set $I_L(j,q)$, $\forall j \in J, \forall q \in bin(j)$; and M' refers to the number of data points in the set $I_R(j,q)$, $\forall j \in J, \forall q \in bin(j)$. Note that the formal definitions of the notations used in these constraints are presented in Section 2.2. The second set of node-complexity-binary-split constraints relates to the boundary cases for data points that are less than the minimum or greater than the maximum threshold values:

$$\sum_{i \in I_{\max}(j)} z_{it} + M'' a_{jm} \leq M'', \quad \forall t \in T_L, \forall m \in A_L(t), \forall j \in J \tag{3}$$

$$\sum_{i \in I_{\min}(j)} z_{it} + M''' a_{jm} \leq M''', \quad \forall t \in T_L, \forall m \in A_R(t), \forall j \in J \tag{4}$$

where M'' refers to the number of data points in the set of $I_{\max}(j), \forall j \in J$; and M''' refers to the number of data points in the set of $I_{\min}(j), \forall j \in J$.

Regarding constraint (1), when any branch node on the left path is deactivated ($d_m = 0, \forall m \in A_L(t)$), all other variables, including z_{it}, a_{jm}, and b_{mr} for $\forall t \in T_L, \forall m \in A_L(t), \forall j \in J$, are set to 0. This means that none of the data points within their bin range will be assigned to any leaf node, and none of the feature thresholds in its bin range will be selected, thereby preventing the classification of data points.

In the case where the branch node is active ($d_m = 1, \forall m \in A_L(t)$), if none of the features are selected ($a_{jm} = 0, \forall m \in A_L(t), \forall j \in J$), the use of a large constant M ensures that the constraint is always satisfied, regardless of the values taken by other variables. However, when features are selected ($a_{jm} = 1, \forall m \in A_L(t), \forall j \in J$) and the condition $\sum_{r \in B_L(j,q)} b_{mr} = 1, \forall m \in A_L(t)$ is met, data points with features falling within the "left" set of $q \in bin(j)$ are constrained to 0. Failure to meet this condition allows data points in that set to potentially take a value of 1.

Similarly, for constraint (2), if no features are selected ($a_{jm} = 0, \forall m \in A_L(t), \forall j \in J$), a large constant M' ensures the constraint is always satisfied. On the other hand, when features are selected ($a_{jm} = 1, \forall m \in A_L(t), \forall j \in J$) and the condition $\sum_{r \in B_R(j,q)} b_{mr} = 0, \forall m \in A_L(t)$ is satisfied, data points with features in the "right" set of $q \in bin(j)$ are forced to be 0. Failure to meet this condition allows data points in that set to potentially take a value of 1.

For constraints (3) and (4), when $a_{jm} = 1, \forall m \in A_L(t)$ or $m \in A_R(t), \forall j \in J$, any data point in the minimum or maximum set of feature j is forced to take a value of 0.

Next, we describe and illustrate each of these constraints further:

- Constraint (1) represents that binary feature selection variables a_{jm} and binary encoding threshold variables b_{mr} in branch nodes $m \in A_L(t)$ will be used to determine whether or not data points land into a leaf node $t \in T_L$ by following the left-direction of its ancestor (a branch node) $m \in A_L(t)$. The sets $I_L(j,q)$ and $B_L(j,q)$ have similar definitions to $I_R(j,q)$ and $B_R(j,q)$, but they depend on the "left" ranges of $bin(j)$. In the above simple example, the "left" ranges of $bin(j)$ are [3.5, 5.5], [5.5, 9.5], and [8.5, 9.5]. When $q = [3.5, 5.5]$, $I_L(j,q) = \{4,5\}$ and $B_L(j,q) = \{1,2\}$.
- Note that we multiply the binary node branching variable d_m at the right-hand-side of the constraint (1) to take the node complexity into account. When a branch node is off (i.e., $d_m = 0$), there is no split rule (i.e., $a_{jm} = 0$ and $b_{mr} = 0$) such that the constraint (1) will force z_{it} to be all zeros. That is, data points would never traverse the left-direction path of the branch node and they will be forced to land at the right-most leaf node by other constraints in the BNP-OCT formulation. Fig. 3 shows an example that when a branch node 2 is off, the data points will follow the path $1 \to 2 \to 5$, and in fact, the active path should be $1 \to 5$ as shown in the right side of the tree structure.
- Similarly, in constraint (2), binary feature selection variables a_{jm} and binary encoding threshold variables b_{mr} in branch nodes $m \in A_R(t)$ will be used to determine whether or not data points land into a leaf node $t \in T_L$ by following the right-direction of its ancestor (a branch node) $m \in A_R(t)$. Also, $I_R(j,q)$ refers to the set of indices of data points whose feature j's values are in the "right" range of $q \in bin(j)$. $B_R(j,q)$ refers to the set of indices of binary encoding variables for feature $j \in J$ and the "right" range of $q \in bin(j)$. In the above simple example, the "right" ranges of $bin(j)$ are [3.5, 5.5], [3.5, 8.5], and [8.5, 9.5]. When $q = [3.5, 5.5]$, $I_R(j,q) = \{4,5\}$ and $B_R(j,q) = \{2\}$.
- Constraint (3) represents that binary feature selection variable a_{jm} will be used to control whether or not data points land into a leaf node $t \in T_L$ by following the left-direction of its ancestor (a branch node) $m \in A_R(t)$. $I_{\max}(j)$ refers to the set of indices of data points whose feature j's values are greater than or equal to the maximum value of all possible thresholds of feature j.
- Similarly, constraint (4) represents that binary feature selection variable a_{jm} will be used to control whether or not data points land into a leaf node $t \in T_L$ by following the right-direction of its ancestor (a branch node) $m \in A_R(t)$. $I_{\min}(j)$ refers to the set of indices of data points whose feature j's values are less than or equal to the minimum value of all possible thresholds of feature j.

In general, constraints (2) and (3) are aimed to utilize the splits associated decision variables (a_{jt} and b_{tr}) to control left or right direction paths followed by data points. Therefore, we formulate the BNP-OCT model by incorporating constraints (2) and (3) into M-OCT model. Our proposed BNP-OCT model extends the methodologies of both M-OCT and BinOCT. Our BNP-OCT model leverages the binary decision

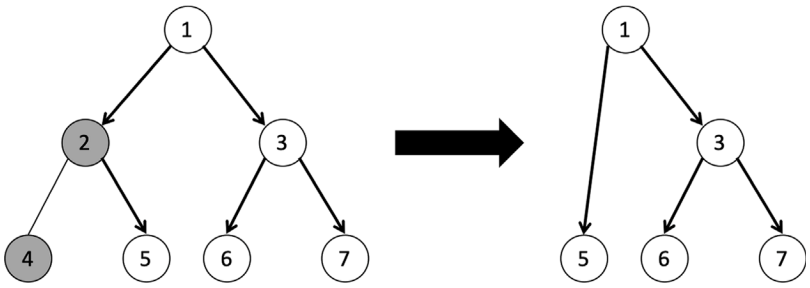


Fig. 3. Tree structures when branch node 2 is turned off.

variables established in BinOCT and introduces node complexity constraints to mitigate overfitting. Additionally, our model incorporates the leaf-branch-interaction constraints from M-OCT, serving to prevent the emergence of invalid tree solutions. This integrated framework enhances the robustness and reliability of the proposed BNP-OCT model. The notation and formulation of BNP-OCT are presented in Section 2.2 and Section 2.3.

2.2. Additional notation

In this section, we present the additional notation for sets, parameters, and variables used in our BNP-OCT formulation.

Sets:

I: the set of data points' indices. $I = \{1, \dots, N\}$.

J: the set of data features' indices. $J = \{1, \dots, F\}$.

K: the set of classes' indices. $K = \{1, \dots, C\}$.

R: the set of binary encoding variables' indices. $R = \{1, \dots, |R|\}$.

T_B: the set of branch nodes in a full tree given the maximum depth $D_{\max}$. $T_B = \{1, \dots, \lfloor T/2 \rfloor\}$, where $T = 2^{D_{\max}+1} - 1$ and it represents the total number of nodes.

T_L: the set of leaf nodes in a full tree given the maximum depth $D_{\max}$. $T_L = \{\lfloor T/2 \rfloor + 1, \dots, T\}$

$A_R(t)$: the set of branch nodes that are the ancestors of leaf node $t \in T_L$ whose right branch has been followed on the path from the root branch node to leaf node t.

$A_L(t)$: the set of branch nodes that are the ancestors of leaf node $t \in T_L$ whose left branch has been followed on the path from the root branch node to leaf node t.

$p(t)$: node t's parent node.

$C_A(t)$: node t's all child leaf nodes except for the right-most one.

$bin(j)$: the set of binary encoding ranges' indices for feature $j \in J$.

$I_R(j, q)$: the set of indices of data points whose feature j's values are in the "right" range of $q \in bin(j)$.

$I_L(j, q)$: the set of indices of data points whose feature j's values are in the "left" range of $q \in bin(j)$.

$B_R(j, q)$: the set of indices of binary encoding variables for feature $j \in J$ and the "right" range of $q \in bin(j)$.

$B_L(j, q)$: the set of indices of binary encoding variables for feature $j \in J$ and the "left" range of $q \in bin(j)$.

$I_{\min}(j)$: the set of indices of data points whose feature j's values are less than or equal to the minimum value of all possible thresholds of feature j.

$I_{\max}(j)$: the set of indices of data points whose feature j's values are greater than or equal to the maximum value of all possible thresholds of feature j.

Parameters:

x_{ij}: the feature j's value of data point i.

y_i: the class label of data point i.

Y_{ik}: 1 if $y_i = k$, otherwise -1.

$\hat{L}$: misclassification counts by simply predicting the most popular class in the entire training data set.

α: complexity parameter from 0 to 1.

$D_{\max}$: the maximum depth of the decision tree.

N: the total number of data points.

$N_{\min}$: the minimum number of data points in leaf nodes.

Variables:

a_{jt}: binary variable, whether or not feature $j \in J$ is selected to split at branch node $t \in T_B$.

b_{tr}: binary variable, binary encoding of threshold variables at branch node $t \in T_B, \forall r \in R$.

d_t: binary variable, whether or not making a split at branch node $t \in T_B$.

l_t: binary variable, whether or not leaf node $t \in T_L$ contains any data points.

c_{kt}: binary variable, indicating if class label $k \in K$ is assigned to leaf node $t \in T_L$.

z_{it}: non-negative variable, indicating if data point $i \in I$ is assigned to leaf node $t \in T_L$.

L_t: non-negative variable, the misclassification counts at leaf node $t \in T_L$.

N_{kt}: non-negative variable, the number of data points with class label k at leaf node $t \in T_L$.

N_t: non-negative variable, the total number of data points at leaf node $t \in T_L$.

2.3. BinNodePenalty-OCT formulation

Based on the above definitions, we develop the following MILP model for the classification tree problem:

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t \tag{5}$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - N \cdot (1 - c_{kt}), \quad \forall k \in K, \forall t \in T_L \tag{6}$$

$$L_t \leq N_t - N_{kt} + N \cdot c_{kt}, \quad \forall k \in K, \forall t \in T_L \tag{7}$$

$$N_{kt} = \frac{1}{2} \sum_{i \in I} (1 + Y_{ik}) z_{it}, \quad \forall k \in K, \forall t \in T_L \tag{8}$$

$$N_t = \sum_{i \in I} z_{it}, \quad \forall t \in T_L \tag{9}$$

$$\sum_{j \in J} a_{jt} = d_t, \quad \forall t \in T_B \tag{10}$$

$$b_{tr} \le d_t, \quad \forall t \in T_B, \forall r \in R \tag{11}$$

$$\sum_{k \in K} c_{kt} = l_t, \quad \forall t \in T_L \tag{12}$$

$$\sum_{i \in T_L} z_{it} = 1, \quad \forall t \in T_L \tag{13}$$

$$z_{it} \le l_t, \quad \forall i \in I, \forall t \in T_L \tag{14}$$

$$\sum_{i \in I} z_{it} \ge N_{min} \cdot l_t, \quad \forall t \in T_L \tag{15}$$

$$d_t \le d_{p(t)}, \quad \forall t \in T_B \setminus \{1\} \tag{16}$$

$$l_t \ge d_{p(t)}, \quad \forall t \in T_L \tag{17}$$

$$\sum_{m \in C_A(t)} l_m \le |C_A(t)| \cdot d_t, \quad \forall t \in T_B \setminus \{1\} \tag{18}$$

$$\sum_{i \in I_L(j,q)} z_{it} + M a_{jm} + M \sum_{r \in B_L(j,q)} b_{mr} \le \left(M + \sum_{r \in B_L(j,q)} M \right) \cdot d_m, \quad \forall t \in T_L, \forall m \in A_L(t), \forall j \in J, \forall q \in bin(j) \tag{19}$$

$$\sum_{i \in I_R(j,q)} z_{it} + M' a_{jm} - M' \sum_{r \in B_R(j,q)} b_{mr} \le M', \quad \forall t \in T_L, \forall m \in A_R(t), \forall j \in J, \forall q \in bin(j) \tag{20}$$

$$\sum_{i \in I_{\max}(j)} z_{it} + M'' a_{jm} \le M'', \quad \forall t \in T_L, \forall m \in A_L(t), \forall j \in J \tag{21}$$

$$\sum_{i \in I_{\min}(j)} z_{it} + M''' a_{jm} \le M''', \quad \forall t \in T_L, \forall m \in A_R(t), \forall j \in J \tag{22}$$

$$L_t \ge 0, \quad \forall t \in T_L \tag{23}$$

$$z_{it} \ge 0, \quad \forall i \in I, \forall t \in T_L \tag{24}$$

$$l_t, c_{kt} \in \{0,1\}, \quad \forall k \in K, \forall t \in T_L \tag{25}$$

$$a_{jt}, b_{tr}, d_t \in \{0,1\}, \quad \forall j \in J, \forall r \in R, \forall t \in T_B. \tag{26}$$

The objective function in Eq. (5) balances the model error (i.e., misclassification counts) and the model complexity (i.e., the number of branching nodes). Constraints (6)–(9) enforce to compute the misclassification counts (L_t) in leaf nodes that are equal to the number of data points (N_t) in leaf nodes less the number of data points of the most common class label (N_{kt}). Constraints (10)–(11) guarantee that at most one feature is allowed to split in branch nodes and force no features to split if branch nodes are turned off. Constraint (12) guarantees that only a single class prediction can be chosen in a leaf node if it contains data points. Constraint (13) forces each data point to reach exactly one leaf node. Constraints (14)–(15) guarantee a minimum number of data points in leaf nodes. Constraint (16) forces a branch node to be off if its parent branch node is off. Constraints (17)–(18) are the proposed "leaf-branch-interaction" constraints to stabilize a valid tree structure. Constraints (19)–(22) are split constraints to control the left or right direction path followed by data points (see Section 2.1 for more details). The rest of the constraints represent the domain of the variables.

3. Valid tree models

In the practical implementation of a classification tree, it is crucial to validate the integrity of the tree. Validity is achieved when the classification results adhere to the specified splitting rules of the tree. If a branch node is designated to split at a certain value, data points with feature values less than or equal to this value must traverse the left direction, while those greater than this value must take the right direction. When a branch node is deactivated, indicating the cessation of dataset splitting, no data points should undergo further division. Adhering to these splitting rules ensures that the classification results align with expectations. Failure to observe these rules may result in misalignment between the intended rules and the actual classification outcomes. While the results may appear satisfactory, applying the rules to test samples or other scenarios could yield significantly erroneous classification results. Such inaccuracies may lead individuals to adopt an inappropriate model, potentially causing complications. Our experimentation revealed that the formulation presented in Bertsimas and Dunn (2017) could generate a tree that appears satisfactory but is, in fact, invalid. We term such trees as "invalid trees". To address this issue, we introduce the leaf-branch-interaction constraint. This constraint aims to ensure that the tree consistently operates as expected, preventing users from being misled into utilizing an erroneous tree based on deceptive results. In this section, we discuss the challenges of the original OCT formulation proposed by Bertsimas and Dunn (2017) and the concept of a valid tree model. The discussion is used to motivate the novel leaf-branch constraints.

Definition 1. A tree solution is "valid" if and only if for every turned-off branch node, all its child leaf nodes except for the right-most one are assigned zero data points.

For example, in Fig. 4(a) the tree is invalid because points are assigned to leaf node 6, but the branch node 3 is turned off (note that nodes 6 and 7 are the child leaf nodes of node 3, and node 7 is the right-most child of node 3). Fig. 4(b) represents a valid tree. The importance of this definition relates to the possibility that an invalid tree (like Fig. 4(a)) might spuriously evaluate to have no classification errors and a desirable parsimony value. Such an evaluation would lead to an undesirable (and nonphysical) tree model solution.

As we discussed in Section 2.1, the previously included split constraints are still key to controlling how data points select left or right directional paths from branch nodes to leaf nodes. These constraints are important because they will form a tree structure as shown in Fig. 3. These split constraints in the previous OCT formulation (Bertsimas and Dunn, 2017) are shown below:

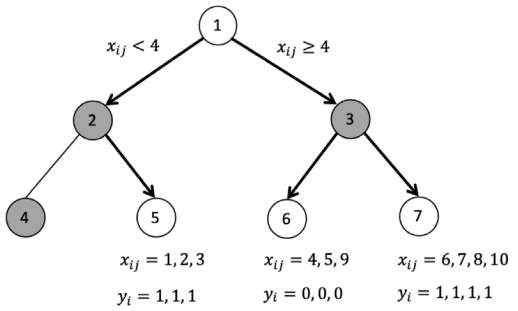
$$\sum_{j \in J} (x_{ij} + \epsilon_j) a_{jm} \le b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t) \tag{27}$$

$$\sum_{j \in J} x_{ij} a_{jm} \ge b_m - (1 - z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_R(t) \tag{28}$$

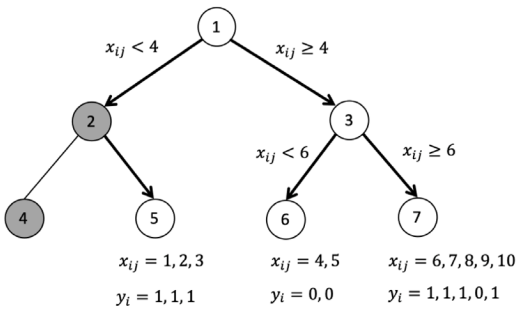
where ϵ_j refers to the smallest non-zero difference among adjacent values of feature j and $\epsilon_{\max} = \max(\epsilon_j)$. Recall that x_{ij} refers to the feature j's value of data point i, a_{jm} is a binary variable indicating whether or not feature j is selected to split in a branch node m, b_m is a threshold variable determining a numerical value for the feature to split at branch node m. The set $A_L(t)$ are the leaf node t's ancestors (branch nodes) whose left branch has been followed on the path from the root branch node to the leaf node t. Similarly, $A_R(t)$ are those right-branch ancestors of the leaf node t.

3.1. An invalid tree example and discussions

To show that the constraints (27) and (28) cannot ensure valid tree solutions in the original OCT formulation, we only need to find an example when they fail. Consider the tree in Fig. 4. This tree involves $N = 10$ data points, $F = 1$ feature, $C = 2$ possible class labels, i.e., $x_{ij} = \{1,2,3,4,5,6,7,8,9,10\}$, and $y_i = \{1,1,1,0,0,1,1,1,0,1\}$. We aim to find a classification tree of maximum depth $D_{\max} = 2$ to minimize the total misclassification counts and the total number of nodes for branching. The optimal solution to the original OCT problem is depicted in Fig. 4(a). That is, only branch node 1 is active while branch nodes 2 and 3 are off. Data points $x_{ij} = \{1,2,3\}$, $y_i = \{1,1,1\}$ are assigned into leaf node 5, data points $x_{ij} = \{4,5,9\}$, $y_i = \{0,0,0\}$ are assigned into leaf node 6, and data points $x_{ij} = \{6,7,8,10\}$, $y_i = \{1,1,1,1\}$ are assigned into leaf node 7. Thus, there is no misclassification error and only one branching node. However, this solution does not form a valid tree structure because when branch node 3 is off the path $1 \to 3 \to 6$



(a) The optimal solution (invalid tree solution) obtained from the OCT formulation.



(b) The optimal solution (valid tree solution) obtained from the M-OCT formulation with leaf-branch-interaction constraints.

Fig. 4. The optimal solutions obtained from the OCT formulation and the proposed M-OCT formulation.

is not available such that data points $x_{ij} = \{4, 5, 9\}$ are not supposed to be in leaf node 6.

Essentially, if a branch node is off, then there is no split rule such that $a = 0$ and $b = 0$, which makes constraint (27) and (28) be able to allow any data points to follow the paths (any left or right direction) from the root node to leaf nodes (these constraints would have $0 \geq 0$ and $0 \leq 0$) such that misclassification counts are zero.

Note that the constraint (27) was the proposed ϵ strategy to overcome the numerical instabilities for the original constraints in the MIO solver as stated in Bertsimas and Dunn (2017). As discussed above, we show that this kind of ϵ strategy can lead to an invalid tree structure and spurious error-free. Clearly, an alternative ϵ strategy is to take the **epsilon** values out of the brackets as shown below:

$$\sum_{j \in J} x_{ij} a_{jm} + \epsilon \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t) \quad (29)$$

where ϵ value could be a very small number or equal to $\epsilon_{\min} = \min(\epsilon_j)$. However, if ϵ is too small, the numerical instabilities in MIO solvers could also lead to an invalid tree structure. We will present the numerical experiments in Section 4.2 to demonstrate the risk of different epsilon strategies.

3.2. Leaf-branch-interaction constraints

Seeking to guarantee a valid tree structure even with numerical challenges, we propose the new leaf-branch-interaction constraints (30)–(31) as repeated below:

$$l_t \geq d_{p(t)}, \quad \forall t \in T_L \quad (30)$$

$$\sum_{m \in C_A(t)} l_m \leq |C_A(t)| \cdot d_t, \quad \forall t \in T_B \setminus \{1\} \quad (31)$$

where l_t refers to a binary variable indicating if a leaf node t contains data points and d_t refers to a binary variable indicating if a branch node t is turned on. Recall that $p(t)$ is defined as the node t's parent nodes. Hence, $p(t)$ in constraint (30) refers to the set of branch nodes that are directly connected to the leaf nodes. In addition, $C_A(t)$ represents branch node t's all child leaf nodes except for the right-most one, and $T_B \setminus \{1\}$ represents the set of branch nodes except for the root branch node 1.

Next, we consider both constraints singly. Our intent is to clarify the reason why leaf-branch constraints increase the chances the derived tree will be valid. The following definition of the **general case** describes the possibility that all leaf nodes would be populated.

Definition 2. The **general case** refers to the condition, for a given tree structure assumed to have a depth greater than 1, that all leaf nodes would have points assigned to them assuming that all branch nodes are turned on. (Note that, if some of the branch nodes are turned off, then their child leaf nodes are not supposed to contain any data points except for the right-most leaf nodes.)

To understand the benefits of adding leaf-branch constraints, consider nodal regulation from the parental perspective. Note that the first set of constraints will not prevent data from reaching leaf nodes in the general case. Note also that constraint (30) is sufficient such that for each leaf node $t \in T_L$, if its parent node $p(t)$ (i.e., a branch node) is turned on, then this leaf node must contain data points in the general case.

To see this, consider the general case. In this case, leaf nodes would contain data points unless the corresponding branch nodes are turned off. By definition, $d_{p(t)}$ turns branch nodes on and off. The constraint (30) ensures that the leaf nodes must contain points if the given parent branch node $p(t)$ is turned on.

An additional consideration relates to the second set of proposed constraints and the perspective of the child nodes. It relates to a guarantee that the problems noted in Section 3.1 will not occur. Constraint (31) is sufficient such that for each branch node $t \in T_B \setminus \{1\}$, if it is turned off, then all its child leaf nodes except for the right-most one ($C_A(t)$) cannot contain any data points. This follows because, in the general case, leaf nodes would not contain data points if their parent branch nodes were turned off. By definition, when a branch node t is turned off, the set of constraints (31) becomes to $\sum_{m \in C_A(t)} l_m \leq 0$, which would force $l_m = 0$, i.e., no data points would be assigned to node t's child leaf nodes except for the right-most one.

Intuitively, taking Fig. 4(a) as an example, for the branch node $t = 3$, all its child leaf nodes are $\{6, 7\}$. Then we will have $C_A(t) = \{6\}$ because the right-most leaf node 7 is excluded. When the branch node 3 is off, constraint (31) will have $l_6 \leq 0$, then leaf node $C_A(t) = \{6\}$ is not supposed to contain any data points such that the path $1 \rightarrow 3 \rightarrow 6$ should be infeasible.

The above results establish that the proposed leaf-branch-interaction constraints are critically important for the optimal solution of tree modeling problems. The following results building on specific selections of binary variables permit the derivation of optimal solutions.

By incorporating the new leaf-branch-interaction constraints into the original OCT, we formulate a modified version of the original OCT model called M-OCT, which is presented in the appendix. For the above simple example, as shown in Fig. 4(b), the M-OCT produces the optimal and valid tree solution. That is, there are two split rules at branch nodes 1 and 3. Data points $x_{ij} = \{1, 2, 3\}$ follow the path $1 \rightarrow 2 \rightarrow 5$ to reach leaf node 5, data points $x_{ij} = \{4, 5\}$ follow the path $1 \rightarrow 3 \rightarrow 6$ to reach leaf node 6, and data points $x_{ij} = \{6, 7, 8, 9, 10\}$ follow the path $1 \rightarrow 3 \rightarrow 7$ to reach leaf node 7.

From our example, we know that without the leaf-branch-interaction constraints, the existing constraints cannot guarantee that a tree will be valid, i.e., turning off a node has the undesirable effect — producing an infeasible path from branch nodes to leaf nodes. The proposed leaf-branch-interaction constraints increase the chance of valid tree structures when small epsilons make solvers unstable. The experiment results of different ϵ strategies are given in Section 4.2.

Table 1
The data sets used in the experiments.

Data set	N	F	C
House-vote	232	16	2
Mushroom	5644	22	2
Chess	3196	36	2
Drybean	13 611	16	7
Nursery	12 960	8	3
Soybean	266	35	15
Breast-cancer-wisconsin	682	9	2
Car-evaluation	1728	6	4
German	1000	20	2
Balance-scale	625	4	3
Biodeg	1055	41	2
Glass	214	9	6
Red-wine	1599	11	6
Tic-tac-toe	958	18	2
White-wine	4898	11	7

4. Experiments

In this section, we perform experiments to evaluate the performance of our proposed models on 15 publicly available data sets from the UCI Machine Learning Repository (Lichman, 2013), and compare the results to CART, OCT (Bertsimas and Dunn, 2017), BinOCT (Verwer and Zhang, 2019), and Max-Flow OCT methods (Aghaei et al., 2020). We also describe the data our study of the pipeline magnetic flux leakage with results in Appendix A.3.

4.1. Comparison with standard data sets

We use similar data sets as in Verwer and Zhang (2019) without missing values. All data are standardized 0–1. The information from these data sets is shown in Table 1. The number of data points ranges from 214 to 13,611, the number of features ranges from 4 to 36, and the number of class labels ranges from 2 to 15. We implement the proposed M-OCT and BNP-OCT models using `pyomo` package (Hart et al., 2011, 2017) in the Python programming language. Also, we choose the CART algorithm implemented in `scikit-learn` package (Pedregosa et al., 2011) as the greedy method for comparisons. All the formulations in the comparison table are solved using the optimization solver Gurobi 10.0.2 and they are run on a Windows 10 machine with 16 GB RAM, 2.9 GHz Intel Core i5-9400. Additionally, as stated in Bertsimas and Dunn (2017) and Verwer and Zhang (2019), a warm start solution obtained from CART is able to facilitate finding a better solution for the methods based on optimization.

For the performance evaluation of these methods, we use two metrics, accuracy, and F1 score, for comparisons. We include the metric of F1 score because some data sets are highly imbalanced such that the metric of accuracy might be misleading. For each data set, we conduct 5-fold cross-validation and present the model metrics in the comparison table. For training each data set, the maximum depth $D_{\max}$ of a tree ranges from 1 to 3, the minimum number of samples $N_{\min}$ in leaf nodes is set as 1, and the complexity parameter α is set as 0.01. The settings of the time limit of the Gurobi solver are 5 min, 15 min, and 30 min for $D_{\max} = 1, 2, 3$, respectively.

Our implemented codes for these models and the training and testing data sets used in the experiments are available online at https://github.com/EnhaoLiu/optimaltree and https://github.com/hutengmu123/optimaltree.

4.2. M-OCT vs previous OCT

As discussed in Section 3, we present the experiment results for the previous OCT models with and without the proposed leaf-branch constraints. The three different ϵ strategies considered are as follows:

- ϵ strategy 1: This strategy was utilized in Bertsimas and Dunn (2017), which is the split constraint (27) as presented before. Here we recap it: $\sum_{j\in J}(x_{ij}+\epsilon_j)a_{jm} \le b_m + (1+\epsilon_{\max})(1-z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t)$ where ϵ_j refers to the smallest non-zero difference among adjacent values of feature j in the training data and $\epsilon_{\max} = \max(\epsilon_j)$.
- ϵ strategy 2: This strategy represents an alternative way of taking the epsilon values out of the brackets compared with strategy 1. Then, we have $\sum_{j\in J} x_{ij}a_{jm} + \epsilon_{\min} \le b_m + (1+\epsilon_{\max})(1-z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t)$ where $\epsilon_{\min} = \min(\epsilon_j)$
- ϵ strategy 3: This strategy is similar to strategy 2 but the only difference is that we pre-define the $\epsilon_{\min}$ value as a small number which is 0.0000001 in our experiment. That is, $\sum_{j\in J} x_{ij}a_{jm} + 0.0000001 \le b_m + (1+\epsilon_{\max})(1-z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t)$

In our stability study, results derive from the 15 most popular data sets, also from the UCI Machine Learning Repository (Lichman, 2013), and are shown in Table 1. We generate $N = 200$ random samples for each data set (with replacement) 50 times. For each specific sample, we run the solver for the six alternative ϵ strategies (three involving leaf-branch constraints) and count the number of times we observe an invalid tree. For simplicity, we only consider failures that occur within the first 5 min of run time using the solver Gurobi. Some reviewers have been skeptical about the stability issue. Therefore, we provide our codes online at https://github.com/hutengmu123/InvalidTree. The findings here include that the proposed leaf-branch constraints eliminate the possibility of invalid trees regardless of the specific ϵ strategy. Further, without the leaf-branch constraints, invalid trees were produced by every ϵ strategy. Finally, the number of invalid trees generated for each ϵ strategy depends on the specific data set with the dry-bean data set being the most challenging, presumably because some values are, after scaling, too close together leading to small ϵ values.

In this experimental investigation, nonzero ϵ values were intentionally introduced into the formulation utilizing variable strategies 1 and 3. The findings, as presented in Table 2, indicate that errors arising from different ϵ values consistently result in the generation of invalid tree structures when employing the OCT formulation outlined in Bertsimas and Dunn (2017). Furthermore, the presence of these invalid structures is not limited to scenarios with formulation errors alone. They also manifest when realistic ϵ values are excessively small as in the case of ϵ strategy 2. The incorporation of leaf-branch constraints serves to stabilize the formulation, effectively mitigating the occurrence of invalid tree structures induced by either small ϵ values or formulation errors. This observation underscores the significant utility of leaf-branch constraints in addressing errors stemming from the use of small ϵ values. Note that the Max-Flow OCT formulation also includes node penalty and achieves consistently valid trees (not shown for space reasons). As we show next, our proposed formulations offer accuracy advantages over Max-Flow OCT, particularly in cases with continuous inputs.

4.3. OCT, BNP-OCT, max-flow OCT, BinOCT vs CART

Table 3 shows that the performance of OCT, BinOCT, Max-Flow OCT, BNP-OCT, and CART approach are comparable for learning all tree depths though BNP-OCT slightly outperforms alternatives for all the cases. Although most of the data instances show relatively comparable accuracy and F1 scores across these four models, OCT and BNP-OCT models have about 53%–116% improvements in F1 scores for the soybean data set compared with CART. Moreover, BinOCT and Max-Flow OCT show an increase in accuracy and F1 score for 50% data sets with max depth equal to 2 and 30% data sets with max depth equal to 3. Relatively large improvements can be observed in glass data set. Based on the analysis presented in Figs. 5 and 6, we illustrate a boxplot depicting the performance improvement relative to CART. Our findings indicate a notable advantage of BNP-OCT, particularly evident

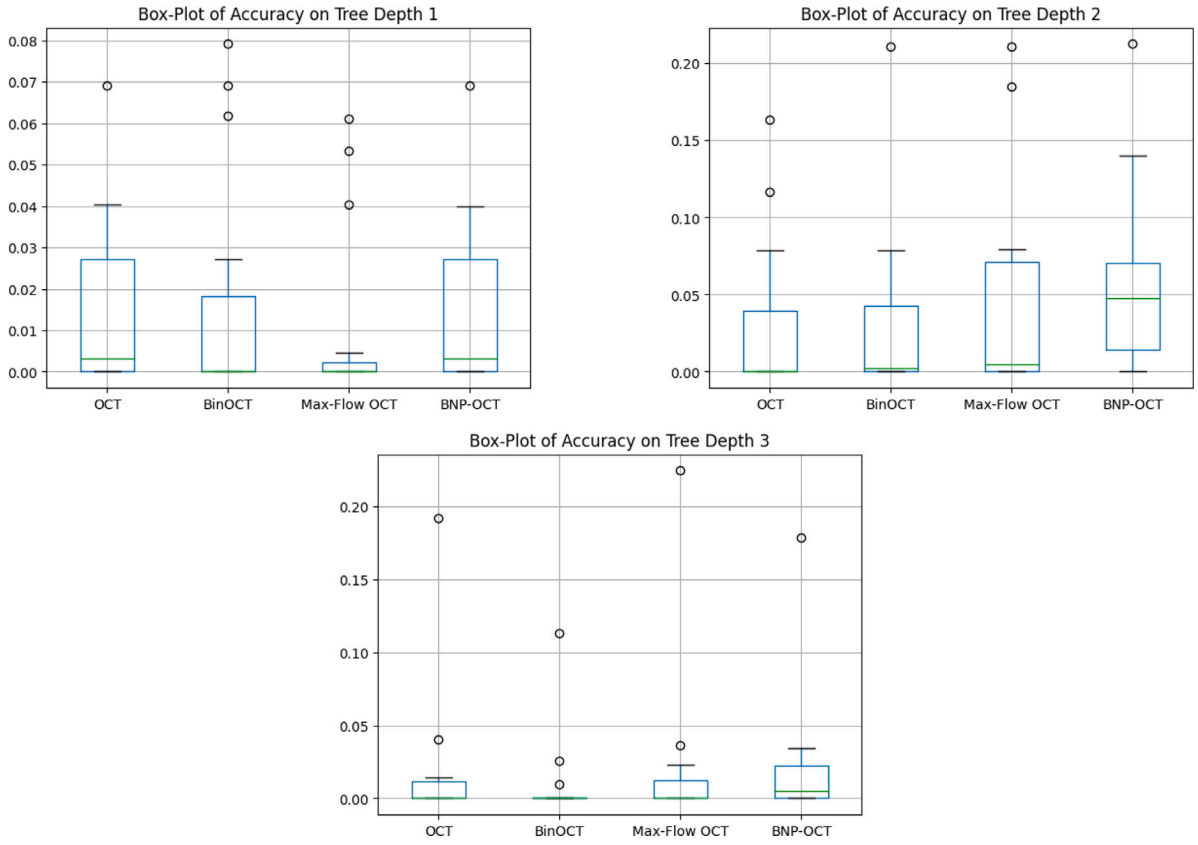


Fig. 5. The boxplot comparison centered on the improvement over CART concerning testing accuracy across tree depths of 1, 2, and 3.

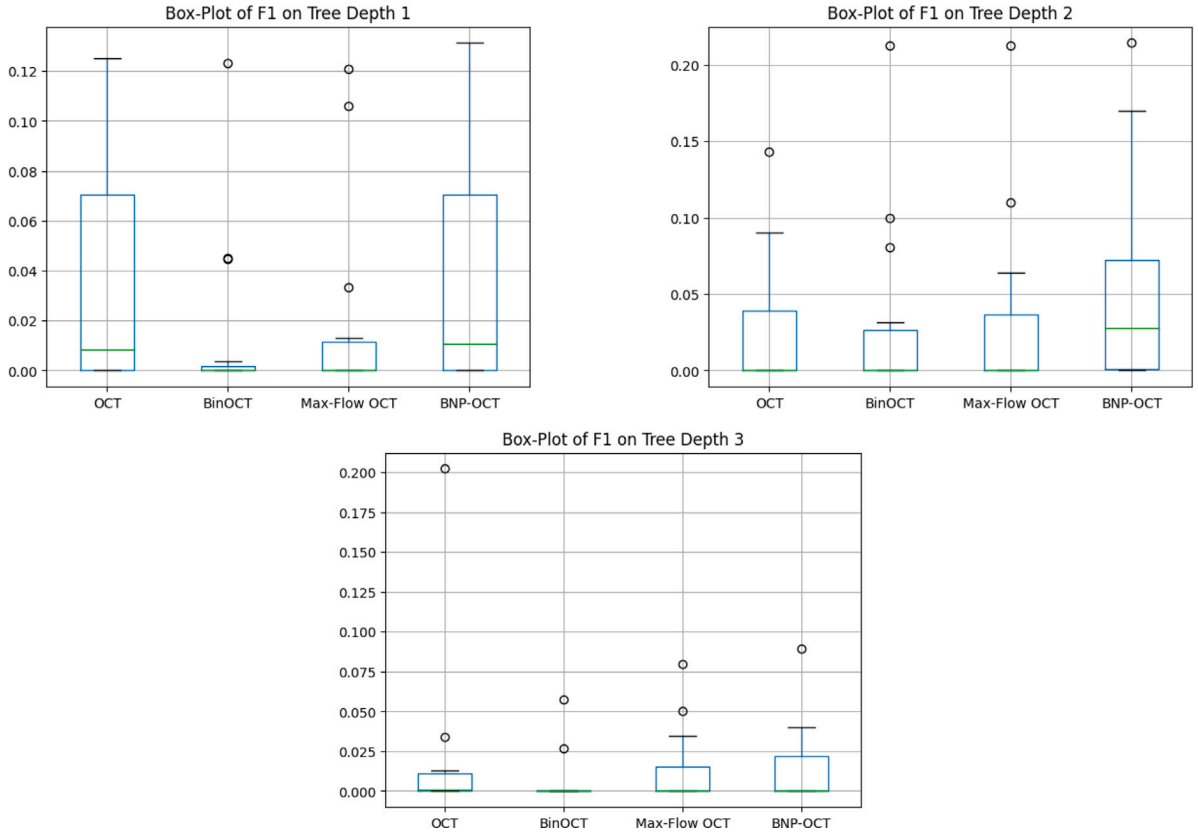


Fig. 6. The boxplot comparison centered on the improvement over CART concerning testing F1 score across tree depths of 1, 2, and 3.

Table 2
The study of stability relating to ϵ strategy with the presence of leaf-branch constraints with D_{max} = 3,4,5. With the leaf-branch constraints, all the invalid trees are eliminated. Without the leaf-branch constraints, every strategy has invalid tree structures. #1, #2 and #3 are the ϵ strategies without leaf-branch constraints. And M-OCT is the OCT method with leaf-branch constraints.

Data set	No. Samples	#1	#2	#3	#1	#2	#3	#1	#2	#3	M-OCT
		depth = 3, time = 5 min			depth = 4, time = 5 min			depth = 5, time = 5 min			All cases
House-vote	50	50	0	50	50	0	50	50	0	50	0
Mushroom	50	50	1	50	50	0	50	50	0	50	0
Chess	50	50	2	50	50	1	50	50	1	50	0
Drybean	50	50	14	50	50	14	50	50	4	50	0
Nursery	50	50	1	50	50	1	50	50	1	50	0
Soybean	50	50	0	50	50	0	50	50	0	50	0
Breast-cancer-wisconsin	50	50	0	50	50	0	50	50	0	50	0
Car-evaluation	50	50	0	50	50	0	50	50	0	50	0
German	50	50	3	50	50	0	50	50	0	50	0
Balance-scale	50	50	1	50	50	0	50	50	1	50	0
Biodeg	50	50	0	50	50	0	50	50	0	50	0
Glass	50	50	2	50	50	1	50	50	0	50	0
Red-wine	50	50	2	50	50	0	50	50	2	50	0
Tic-tac-toe	50	50	1	50	50	0	50	50	0	50	0
White-wine	50	50	2	50	50	1	50	50	1	50	0

Table 3
Evaluation comparisons on testing data sets: the average accuracy and F1 scores of CART, OCT, BinOCT, Max-Flow OCT, and BNP-OCT with warm starts using 5-fold cross-validation. The best-performing model is marked in bold.

Data set	Metric	CART	OCT	BinOCT	Max-Flow OCT	BNP-OCT	CART	OCT	BinOCT	Max-Flow OCT	BNP-OCT	CART	OCT	BINOCT	Max-Flow OCT	BNP-OCT
		max depth = 1, time limit = 5 min					max depth = 2, time limit = 15 min					max depth = 3, time limit = 30 min				
House-vote	Accuracy	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	**0.9698**	0.9481	**0.9611**	0.9482	0.9395	**0.9611**
Mushroom	Accuracy	**0.9390**	0.9345	0.3367	0.6821	0.9357	0.9390	0.9390	0.6253	0.7744	**0.9868**	**0.9872**	0.8330	0.4611	0.8179	0.8795
Chess	Accuracy	0.3267	**0.3268**	**0.3268**	**0.3268**	**0.3268**	0.5277	0.6443	0.7384	0.7384	**0.7402**	0.9064	0.6997	**0.9158**	0.8317	0.9064
Drybean	Accuracy	0.0605	0.0605	0.0605	0.0605	**0.0693**	0.0785	0.0605	0.1329	**0.1577**	0.1333	0.1399	0.0605	0.1355	0.1329	**0.1654**
Nursery	Accuracy	0.0001	**0.0693**	**0.0693**	0.0001	**0.0693**	0.0001	0.0001	0.0233	0.0001	**0.1333**	0.0001	0.0001	0.0256	0.0001	**0.0304**
Soybean	Accuracy	0.0151	**0.0245**	**0.0245**	0.0196	0.0001	0.0604	0.0604	0.0624	0.0653	**0.0755**	0.1882	0.1652	0.1533	0.1658	**0.1769**
Breast-cancer-wisconsin	Accuracy	0.8974	**0.9062**	0.8679	0.8834	**0.9062**	0.9282	0.9224	0.8724	0.8957	**0.9458**	0.9312	0.9370	0.8444	0.8729	**0.9414**
Car-evaluation	Accuracy	**0.6999**	**0.6999**	**0.6999**	**0.6999**	**0.6999**	**0.7777**	0.7325	**0.7777**	0.7654	0.7325	0.7632	0.7354	0.7452	**0.7695**	0.7354
German	Accuracy	0.7000	**0.7170**	0.4930	0.6328	**0.7170**	0.6910	0.6950	0.5900	0.6566	**0.7040**	0.7200	0.6880	0.5310	0.5764	**0.7250**
Balance-scale	Accuracy	0.5328	0.5008	**0.5600**	0.5328	0.5008	0.5888	**0.6320**	0.6432	**0.6320**	0.6160	0.6544	0.6688	0.6368	0.6556	**0.6734**
Biodeg	Accuracy	0.6028	0.6427	**0.6645**	0.6560	0.6417	0.5886	0.6246	**0.6673**	0.6528	0.6635	0.7791	0.7498	0.7640	0.7369	**0.7735**
Glass	Accuracy	0.0744	0.1116	**0.1535**	0.1355	0.1116	0.1116	0.2748	0.1399	**0.2965**	0.2515	0.1333	0.3257	0.2467	**0.3578**	0.3117
Red-wine	Accuracy	0.5447	**0.5479**	0.3978	0.3851	**0.5479**	0.5072	0.5304	0.4009	0.5322	**0.5633**	0.5391	0.5460	0.2344	**0.5754**	0.5385
Tic-tac-toe	Accuracy	**0.5128**	**0.5128**	0.4044	0.5077	**0.5128**	0.5138	0.4464	**0.5451**	0.5055	0.5089	0.4442	**0.4849**	0.3273	0.4674	0.4786
White-wine	Accuracy	0.4471	**0.4874**	0.4490	**0.4874**	0.4871	0.4471	0.5257	0.3266	**0.5258**	0.5128	0.5117	0.5217	0.3759	**0.5306**	0.5127
House-vote	F1	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9697**	0.9481	**0.9610**	0.9480	0.9392	**0.9610**
Mushroom	F1	0.9209	0.9209	0.2768	0.7270	**0.9343**	0.9862	**0.9868**	0.5843	0.7921	**0.9868**	**0.9887**	0.8353	0.3993	0.8388	0.8955
Chess	F1	**0.3177**	0.3045	**0.3177**	**0.3177**	**0.3177**	0.5541	0.6445	0.7663	0.7663	**0.7682**	**0.9332**	0.6985	0.9292	0.8839	**0.9332**
Drybean	F1	0.0406	0.0447	0.0443	**0.0495**	0.0445	0.0650	0.0554	0.0862	**0.0945**	0.0658	0.1707	0.0299	0.1975	**0.2211**	0.2012
Nursery	F1	0.0001	**0.0890**	0.0001	0.0001	**0.0890**	0.0001	0.0001	0.0001	0.0001	**0.1333**	0.0001	0.0001	0.0001	0.0001	**0.0399**
Soybean	F1	0.0046	0.1297	0.1277	0.1256	**0.1358**	0.0239	0.1668	0.1237	0.1335	**0.1934**	0.1744	**0.3766**	0.2321	0.2544	0.2637
Breast-cancer-wisconsin	F1	0.8982	**0.9064**	0.8723	0.8652	**0.9064**	0.9282	0.9214	0.8756	**0.9500**	0.9456	0.9298	0.9373	0.8494	**0.9493**	0.9416
Car-evaluation	F1	**0.5816**	**0.5816**	**0.5816**	**0.5816**	**0.5816**	**0.7710**	0.7167	**0.7710**	0.6985	0.7167	**0.7486**	0.7264	0.7263	0.7166	0.7264
German	F1	0.5768	**0.6592**	0.4920	0.5765	**0.6592**	0.6157	**0.6575**	0.5543	0.5765	0.6504	**0.6846**	0.6485	0.5167	0.5765	0.6439
Balance-scale	F1	0.4472	0.4403	**0.4922**	0.4565	0.4403	0.5516	0.6204	**0.6319**	0.6155	0.6045	0.6479	**0.6490**	0.6234	0.6455	0.6352
Biodeg	F1	0.6421	**0.7003**	0.6119	0.6552	**0.7003**	0.6462	0.6830	0.6270	0.6897	**0.7373**	0.8364	0.8100	0.7754	0.7658	**0.8318**
Glass	F1	0.0411	0.0744	**0.0859**	0.0744	0.0744	**0.3077**	0.3034	0.1654	0.2556	0.2894	0.3726	**0.3847**	0.2379	0.3255	0.3679
Red-wine	F1	0.4784	**0.4890**	0.2316	0.2363	**0.4890**	0.4646	0.4875	0.2345	0.3276	**0.4926**	0.4956	0.5053	0.1133	**0.5304**	0.4950
Tic-tac-toe	F1	0.5421	0.5421	0.4627	**0.5523**	0.5421	0.5642	0.5587	0.5960	0.5883	**0.6062**	0.5412	0.5754	0.4105	0.5525	**0.5801**
White-wine	F1	0.2951	0.4123	0.2799	0.4012	**0.4130**	0.4593	0.4557	0.1729	0.2576	**0.4792**	0.4666	0.4680	0.2350	0.2758	**0.4758**

in trees with depths of 2 and 3. Notably, in data sets characterized by relative imbalance, this advantage is more pronounced when considering the F1 metric compared to accuracy. Specifically, in terms of accuracy at depths 2 and 3, BNP-OCT demonstrates superior performance across key statistical measures including maximum, median, and minimum. Similarly, when evaluating F1 scores at depths 2 and 3, BNP-OCT exhibits enhanced performance, characterized by higher maximum values, medians, and wider interquartile ranges.

Given that both OCT and BNP-OCT models are able to solve several data instances optimally within the given time limit for depth from 1 to 3, we then compare their running time on the training data sets shown in Table 4. For those data sets solved optimally in all depths from 1 to 3, the BNP-OCT model is about 5–45 times faster in obtaining optimal solutions than the OCT model. Fig. 7 also presents the upper and lower bound evolution for solving OCT and BNP-OCT models on learning the tree of maximum depth of 3 for the full iris data set. BNP-OCT is able to find the optimal solution in a shorter time (about 20 s) and prove its optimality in 80 s, while OCT spends about 230 s to find the optimal solution and proves its optimality in 330 s. For most of the data sets, BinOCT, BNP-OCT, and OCT do not obtain the optimal solution within the time limit for learning the tree of depth equal to 3. Max-Flow OCT shows advantages in obtaining optimal solutions in a short time for the large tree size but it is sometimes worse than BNP-OCT in either testing accuracy, or tree node complexity.

In addition, when learning a tree of depth equal 1 in those data sets with a small size (less than 200 points), there is no significant difference in run time among all formulation methods. For data sets with relatively large possible thresholds in features (e.g., drybean data set), every formulation method does not find the optimal solutions in a given time. Based on the optimality gaps shown in Table 5, OCT does not find the optimal solution for data sets drybean and nursery and both BinOCT and BNP-OCT do not find the optimal solution for data set drybean because of the size of the problem given max depth equal to 1. In general, Max-Flow OCT is faster compared to other methods for a given depth. For max depth equals 3, OCT, BinOCT, and BNP-OCT have trouble reducing the optimality gaps to 0% but Max-Flow OCT is still able to find optimal solutions for most of the data sets.

In summary, compared to the CART approach, our proposed BNP-OCT models are able to learn more desirable classification trees on testing data sets in terms of accuracy and F1 scores for those balanced and imbalanced data sets. For data sets with fewer possible thresholds in their features, compared to other methods, the BNP-OCT model obtains optimal solutions more efficiently due to using few binary variables and constraints. In Table 6, the metrics of boxplot for all the methods are comparable, and our method BNP-OCT has better testing

Table 4
Evaluation comparisons on the solver running time (seconds) of OCT, BinOCT, Max-Flow OCT, and BNP-OCT. The minimum running time is marked in bold.

Data set	N	F	C	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT
				max depth = 1, time limit = 5 min				max depth = 2, time limit = 15 min				max depth = 3, time limit = 30 min			
House-vote	232	16	2	0.05	0.07	**0.02**	0.03	3.99	1.47	**0.91**	1.37	417.14	maxTime	**3.64**	121.42
Mushroom	5644	22	2	21.47	79.78	**7.46**	2.27	maxTime	maxTime	**218.50**	maxTime	maxTime	maxTime	**879.20**	maxTime
Chess	3196	36	2	24.8	3.56	6.82	**0.78**	maxTime	maxTime	**13.26**	maxTime	maxTime	maxTime	**165.16**	maxTime
Drybean	13 611	16	7	maxTime	maxTime	**4.63**	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime
Nursery	12 960	8	3	maxTime	4.52	31.51	**4.12**	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime	maxTime
Soybean	266	35	15	2.39	0.58	**0.09**	1.31	maxTime	maxTime	**11.83**	54.97	maxTime	maxTime	**356.84**	maxTime
Breast-cancer-wisconsin	682	9	2	0.42	0.14	0.37	**0.10**	393.48	14.26	**3.52**	67.60	maxTime	maxTime	**59.22**	maxTime
Car-evaluation	1728	6	4	3.48	0.44	1.47	**0.21**	maxTime	113.23	**30.14**	112.35	maxTime	maxTime	**36.63**	maxTime
German	1000	20	2	11.63	0.96	1.67	**0.83**	maxTime	maxTime	**38.84**	maxTime	maxTime	maxTime	maxTime	maxTime
Balance-scale	625	4	3	0.93	0.09	0.39	**0.06**	461.10	11.01	**1.95**	85.36	maxTime	maxTime	**126.70**	maxTime
Biodeg	1055	41	2	15.67	14.93	**1.46**	16.81	maxTime	maxTime	**3.46**	maxTime	maxTime	maxTime	**35.04**	maxTime
Glass	214	9	6	1.51	0.51	**0.18**	1.05	maxTime	216.18	**10.87**	379.7	maxTime	maxTime	**256.32**	maxTime
Red-wine	1599	11	6	17.76	3.28	**0.93**	5.97	maxTime	356.20	**42.15**	maxTime	maxTime	maxTime	**578.32**	maxTime
Tic-tac-toe	958	18	2	2.15	0.20	1.40	**0.18**	maxTime	250.11	**83.85**	84.95	maxTime	maxTime	**917.19**	maxTime
White-wine	4898	11	7	141.65	10.18	**2.13**	22.01	maxTime	maxTime	**6.82**	maxTime	**1032.22**	maxTime	maxTime	maxTime

Table 5
Evaluation comparisons on the optimality gaps of OCT, BinOCT, Max-Flow OCT, and BNP-OCT.

Data set	N	F	C	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT
				max depth = 1, time limit = 5 min				max depth = 2, time limit = 15 min				max depth = 3, time limit = 30 min			
House-vote	232	16	3	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	100%	**0%**	**0%**
Mushroom	5644	22	2	**0%**	**0%**	**0%**	**0%**	68.81%	100%	**0%**	53.22%	82.01%	**0%**	**0%**	83.10%
Chess	3196	36	2	**0%**	**0%**	**0%**	**0%**	90.14%	**0%**	**0%**	93.01%	95.81%	100%	**0%**	92.50%
Drybean	13 611	16	7	98.75%	8.21%	**0%**	98.77%	97.93%	99.99%	**52.30%**	98.98%	96.30%	100%	100%	**95.60%**
Nursery	12 960	8	3	69.50%	**0%**	**0%**	**0%**	98.86%	89.96%	**13.50%**	98.46%	98.90%	100%	154.71%	**98.85%**
Soybean	266	35	15	**0%**	**0%**	**0%**	**0%**	97.04%	91.41%	**0%**	**0%**	34.86%	17.31%	**0%**	60.55%
Breast-cancer-wisconsin	682	9	2	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	74.47%	100%	**0%**	64.10%
Car-evaluation	1728	6	4	**0%**	**0%**	**0%**	**0%**	89.22%	**0%**	**0%**	**0%**	96.78%	91.19%	**0%**	94.24%
German	1000	20	2	**0%**	**0%**	**0%**	**0%**	97.83%	99.99%	**0%**	97.77%	100%	**17.35%**	97.81%	96.32%
Balance-scale	625	4	3	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**	85.36%	93.63%	90.61%	**0%**	81.09%
Biodeg	1055	41	2	**0%**	**0%**	**0%**	**0%**	96.78%	100%	**0%**	98.41%	95.95%	100%	**0%**	98.19%
Glass	214	9	6	**0%**	**0%**	**0%**	**0%**	92.47%	**0%**	**0%**	**0%**	96.15%	100%	**0%**	94.22%
Red-wine	1599	11	6	**0%**	**0%**	**0%**	**0%**	98.69%	**0%**	**0%**	97.21%	97.20%	100%	**0%**	98.67%
Tic-tac-toe	958	18	2	**0%**	**0%**	**0%**	**0%**	96.40%	**0%**	**0%**	**0%**	96.20%	100%	**0%**	88.55%
White-wine	4898	11	7	**0%**	**0%**	**0%**	**0%**	98.84%	91.06%	**0%**	98.87%	98.85%	100%	**41.50%**	98.89%

Table 6
Summary statistics of OCT, BinOCT, Max-Flow OCT, and BNP-OCT. The best value is marked in bold.

Data set	Metric	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT	OCT	BinOCT	Max-Flow OCT	BNP-OCT
		max depth = 1, time limit = 5 min				max depth = 2, time limit = 15 min				max depth = 3, time limit = 30 min			
Test Accuracy	Mean	**0.5008**	0.4318	0.4653	0.4997	0.5372	0.5010	0.5445	**0.5691**	0.5585	0.4897	**0.5620**	0.4873
Test Accuracy	Minimum	**0.0245**	**0.0245**	0.0001	0.0001	0.0001	0.0233	0.0001	**0.0755**	0.0001	0.0256	0.0001	**0.0304**
Test Accuracy	Maximum	**0.9698**	**0.9698**	**0.9698**	**0.9698**	0.9698	0.9698	0.9698	**0.9868**	**0.9611**	0.9482	0.9395	**0.9611**
Test Accuracy	First Quartile	0.2192	**0.2402**	0.2312	0.2192	0.3606	0.2333	**0.4010**	0.3802	0.4053	0.2406	**0.4126**	0.3952
Test Accuracy	Medium	**0.5128**	0.4044	0.5077	**0.5128**	0.6246	0.5900	**0.6320**	0.6160	0.6688	0.4611	0.5764	**0.6734**
Test Accuracy	Third Quartile	**0.7085**	0.6123	0.6691	**0.7085**	0.7138	0.7029	0.7519	**0.7364**	0.7426	0.7546	0.7937	**0.8265**
Test F1	Mean	0.4843	0.3898	0.4393	**0.4865**	0.5485	0.4773	0.5144	**0.5759**	0.5737	0.4796	0.5650	**0.5995**
Test F1	Minimum	**0.0447**	0.0001	0.0001	0.0445	0.0001	0.0001	0.0001	**0.0658**	0.0001	0.0001	0.0001	**0.0399**
Test F1	Maximum	**0.9697**	**0.9697**	**0.9697**	**0.9697**	**0.9868**	0.9697	0.9697	**0.9868**	**0.9610**	0.9480	0.9493	**0.9610**
Test F1	First Quartile	0.2171	0.1797	0.1810	**0.2268**	0.3796	0.1692	0.2566	**0.3843**	**0.4264**	0.2336	0.3007	0.4219
Test F1	Medium	**0.4890**	0.3177	0.4565	**0.4890**	**0.6204**	0.5843	0.5883	0.6062	**0.6485**	0.4105	0.5765	0.6352
Test F1	Third Quartile	**0.6798**	0.5369	0.6184	**0.6798**	0.6999	0.6991	0.7324	**0.7528**	0.7682	0.7509	0.8023	**0.8637**
Run Time (seconds)	Average	56.25	27.32	**4.04**	23.72	777.24	544.16	**71.07**	532.42	1656.62	1800	**596.52**	1688.10
Optimality Gap	Average	11.22%	0.55%	**0%**	6.58%	74.87%	44.83%	**4.39%**	54.75%	83.81%	81.10%	**26.28%**	88.53%
Node complexity	Average	1	1	1	1	2.53	3	2.67	**2.33**	5.87	7	5.80	**5.40**

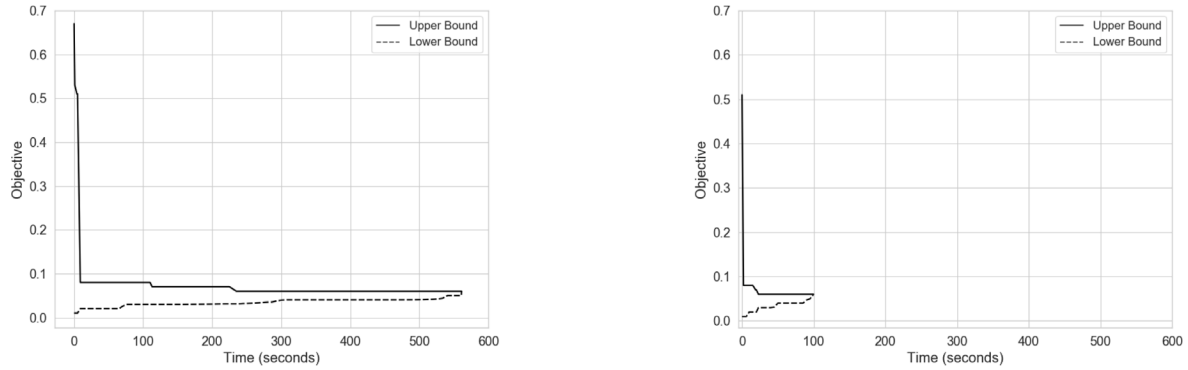
accuracy and F1 score for trees with all depths on average. There are also mixed results for the mean, medium and first quartile of testing accuracy, and each method wins in some cases. The overall fastest method is Max-Flow OCT but its efficiency depends on the structure of the data to achieve the best accuracy in both the training set and testing set. Finally, BNP-OCT compares favorably at reducing the node complexity of the trees whereas BinOCT does not consider the node complexity of the trees.

5. Conclusion

In this article, we propose leaf-branch-interaction and constraints to form the novel M-OCT formulation. We also demonstrate computationally that leaf-branch-interaction constraints eliminate the cases in which solvers will generate "invalid tree" structures. "Validity" is a term that we define to relate to intuitive conditions for tree models. We provide evidence that our leaf-branch-interaction constraints effectively eliminate the chance of generating invalid trees. In practice, the general case does not always occur but we were able to show that stability issues often occur for random subsets of popular data sets leading to invalid trees for all the ϵ strategies that we considered not involving the proposed leaf-branch constraints. In addition, building on the binary encoding thresholds from Verwer and Zhang (2019), we also propose a BNP-OCT model that incorporates leaf-branch-interactions, binary splits, and node complexity constraints. We provide computational results showing that BNP-OCT efficiently and optimally solves decision tree classification problems while taking both model errors and model complexity into account. Significantly, our proposed BNP-OCT model requires relatively few binary decision variables associated with branch and leaf nodes in the learned tree, and the size of these variables is independent of the number of data points.

Through numerical experiments on 15 standard data sets, our proposed M-OCT and BIN-OCT optimization models outperform the widely used greedy approach CART in the testing data sets for all tree depths in terms of model accuracy and F1 score. Specifically, the proposed MILP-based models improve accuracy by 1%–5% and F1 score by 2%–8% for most of the training data sets and above 50% improvements for some specific data sets. In assessing the performance improvement over CART, our research demonstrates that the proposed MILP-based models exhibit superior performance, particularly evident in the F1 scores for trees with depths of 2 and 3 when compared to alternative methods. Relating to speed, although our proposed models are MILP-based approaches that essentially require a longer time to obtain optimal trees, BNP-OCT shows efficiency in learning the optimal trees for data sets with a few possible thresholds in their features because of fewer binary variables and splits constraints used in the model.

The results of our pipeline inspection case study are also impressive. The model generates high levels of sensitivity and specificity in training and test sets. The simplicity and interpretability of the derived model



(a) M-OCT **(b)** BNP-OCT

Fig. 7. The upper and lower bound evolution comparisons for solving M-OCT and BNP-OCT models on learning the tree of maximum depth of 3 for the Iris data set with $N = 150$, $F = 4$, and $C = 3$. (a) M-OCT, (b) BNP-OCT.

are promising for achieving acceptance from the human inspection teams and prediction on unseen pipes.

Many topics remain for future research. First, some data sets involve large numbers of features and also large numbers of possible threshold values. Then, for our proposed BNP-OCT formulation, the total number of constraints can offer a computational challenge. This would result in a large number of rows in the matrix form of the MILP model making it hard for the MIO solver to solve it efficiently. Sampling or selectively including constraints might result in improved efficiency. Column generation might also lead to improvements. Second, constructing a bounding procedure to the original optimal tree problems may improve the computational solutions and bounding efficiencies. Third, it may be desirable to investigate the efficient approaches for selecting representative data points or feature thresholds that effectively reduce the feature space and data sizes. Fourth, since optimal trees are more interpretable and explainable than ensemble learning trees such as XGBoost (Chen and Guestrin, 2016) and RandomForest (Breiman, 2001), our proposed BNP-OCT and M-OCT models could be applied to a wide range of classification problems that require more interpretable and accurate AI models including cybersecurity intrusion risks prediction (Allen et al., 2018; Liu et al., 2019).

In addition, our applied work suggests the importance of additional work relating to model uncertainties. Re-sampling and other methods may clarify probabilities or significance levels associated with feature inclusions and branching values. Clarifying the trust levels associated with tree model features visually is also important for even higher levels of explainability. Also, the uncertainties associated with specific predictions can be clarified in the context of new optimal formulations. Finally, further study of the pipeline investigation is needed. Future investigations can create trustworthy models for unseen pipes and achieve acceptance and integration with human-led inspection teams.

CRediT authorship contribution statement

Enhao Liu: Writing – original draft, Visualization, Methodology, Investigation. **Tengmu Hu:** Writing – review & editing, Visualization, Validation, Software, Investigation. **Theodore T. Allen:** Writing – review & editing, Funding acquisition, Conceptualization. **Christoph Hermes:** Writing – review & editing, Supervision, Data curation.

Data availability

We have shared the link to my data code in the paper.

Acknowledgments

We thank National Science Foundation Grant #1912166 and ROSEN USA, Inc. #109079 for financial support and Rajiv Ramnath, Samantha Krening, Dirk Maiwald, and Marc Fischer for general support for this and related research. Also, Josephine Allen, Cathy Xia, Evelyn Arrey, Maha Yazbeck, Matt Booth, and Christopher Hansen provided valuable feedback.

Appendix

A.1. Modified-OCT formulation

This new formulation takes the original OCT formulation from Bertsimas and Dunn (2017), which did not involve efficient binary splits variables, and augments that formulation to guarantee valid trees with the leaf-branch-interaction constraints (A.13) and (A.14) in the general case. Note that the original OCT formation does not contain these two sets of constraints.

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in T_L} L_t + \alpha \sum_{t \in T_B} d_t \tag{A.1}$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - N \cdot (1 - c_{kt}), \quad \forall k \in K, \forall t \in T_L \tag{A.2}$$

$$L_t \leq N_t - N_{kt} + N \cdot c_{kt}, \quad \forall k \in K, \quad \forall t \in T_L \tag{A.3}$$

$$N_{kt} = \frac{1}{2} \sum_{i \in I} (1 + Y_{ik}) z_{it}, \quad \forall k \in K, \forall t \in T_L \tag{A.4}$$

$$N_t = \sum_{i \in I} z_{it}, \quad \forall t \in T_L \tag{A.5}$$

$$\sum_{j \in J} a_{jt} = d_t, \quad \forall t \in T_B \tag{A.6}$$

$$0 \leq b_t \leq d_t, \quad \forall t \in T_B \tag{A.7}$$

$$\sum_{k \in K} c_{kt} = l_t, \quad \forall t \in T_L \tag{A.8}$$

$$\sum_{t \in T_L} z_{it} = 1, \quad \forall t \in T_L \tag{A.9}$$

$$z_{it} \leq l_t, \quad \forall i \in I, \forall t \in T_L \tag{A.10}$$

$$\sum_{i \in I} z_{it} \geq N_{min} \cdot l_t, \quad \forall t \in T_L \tag{A.11}$$

$$d_t \leq d_{p(t)}, \quad \forall t \in T_B \setminus \{1\} \tag{A.12}$$

$$l_t \geq d_{p(t)}, \quad \forall t \in T_L \tag{A.13}$$

$$\sum_{m \in C_A(t)} l_m \leq |C_A(t)| \cdot d_t, \quad \forall t \in T_B \setminus \{1\} \tag{A.14}$$

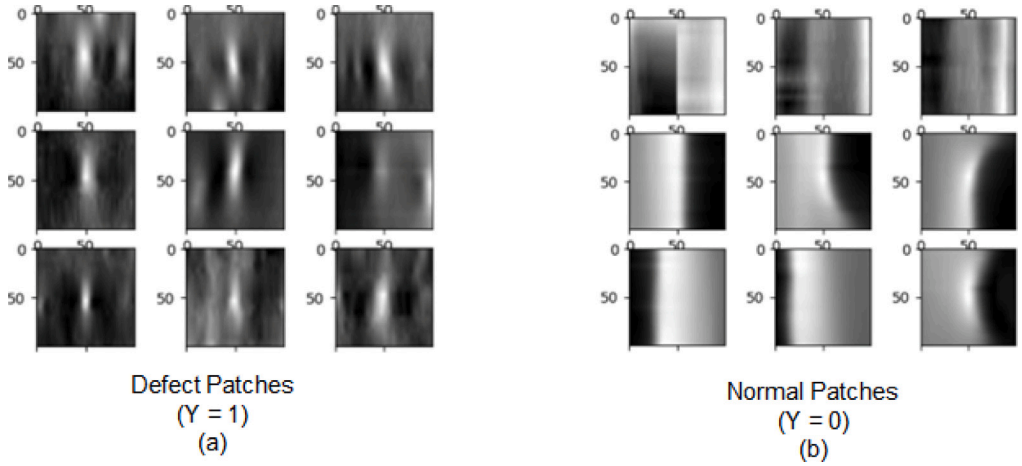


Fig. A.8. Magnetic flux leakage images of pipe sections: (a) defects and (b) non-defects.

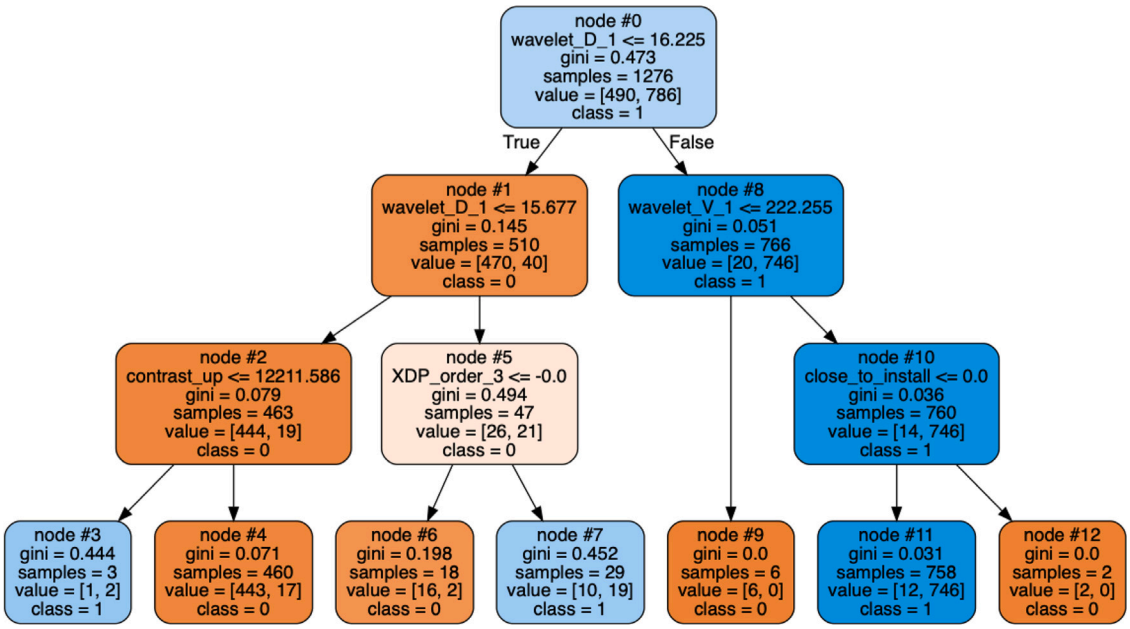


Fig. A.9. Optimal Classification Tree fit to the training set (490 normal patches and 786 defect patches). The model achieves 97.4% precision and 96.4% recall on the training set, and 96.6% precision and 95.8% recall on the unseen testing set (216 normal patches and 331 defect patches).

$$\sum_{j\in J}(x_{ij}+\epsilon_j)a_{jm} \le b_m + (1+\epsilon_{\max})(1-z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_L(t) \tag{A.15}$$

$$\sum_{j\in J} x_{ij}a_{jm} \ge b_m - (1-z_{it}), \quad \forall i \in I, \forall t \in T_L, \forall m \in A_R(t) \tag{A.16}$$

$$L_t \ge 0, \quad \forall t \in T_L \tag{A.17}$$

$$z_{it}, l_t, c_{kt} \in \{0,1\}, \quad \forall i \in I, \forall k \in K, \forall t \in T_L \tag{A.18}$$

$$a_{jt}, d_t \in \{0,1\}, \quad \forall j \in J, \forall t \in T_B \tag{A.19}$$

A.2. Feature generation for pipeline

In this section, we present the details about how to compute the feature values based on MFL sensor data. The original MFL data are numerical values obtained from the Primary Horizontal Hall (PHH) sensor data.

Feature Type A: the following features are generated from each patch (10 cm × 10 cm).

- Saliency Feature (dim = 1)
  For each patch, we first normalize it into 0–1 range and then convert it to 0–255 range.
- Contrast Feature (dim = 1)
- Wavelet Features (dim = 3 * Decomposition Level)
- X-axis Direction Profile (dim = Number of Polynomial Terms)
- Y-axis Direction Profile (dim = Number of Polynomial Terms)
- Indicators - Whether or not Close to Circumferential Part (dim = 1) - Whether or not Close to Installation Part (dim = 1) - Whether or not Close to Bend Part (dim = 1)

Feature Type B: the following features are generated from each patch (100 × 100) and its surrounding regions (30 cm × 30 cm).

- Surrounding Contrast Feature (dim = 1)
- Surrounding Patches' Individual Contrast Feature (dim = 4)
- Surrounding Blob Area Proportion (dim = 1)

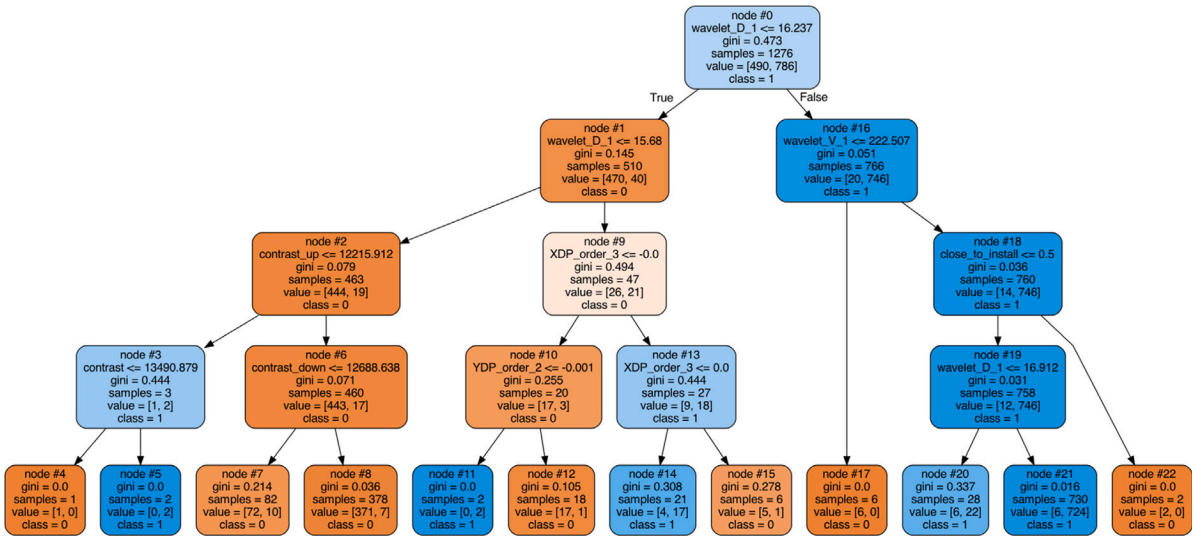


Fig. A.10. CART fit the training set (490 normal patches and 786 defect patches). The model achieves 98.0% precision and 97.6% recall on the training set, and 95.2% precision and 95.8% recall on the unseen testing set (216 normal patches and 331 defect patches).

Table A.7
Feature sets, names, and descriptions.

Feature Set	Feature Names Used in Model	Description
Saliency Feature	Saliency	Saliency score provides how overall salient to be for a given patch.
Contrast Feature	Contrast	Contrast score is a measure of the dissimilarity of an inner window to its immediate surrounding area for a given patch.
Wavelet Features	wavelet_H_1 wavelet_V_1 wavelet_D_1	Frobenius norm of horizontal, vertical, and diagonal detail coefficients at level 1 based on Harr Wavelet Transform.
X-axis Direction Profile (XDP)	XDP_order_1 XDP_order_2 XDP_order_3	XDP feature represents coefficients of the fitted polynomial function in the X-axis direction of a given patch.
Y-axis Direction Profile (YDP)	YDP_order_1 YDP_order_2 YDP_order_3	YDP feature represents coefficients of the fitted polynomial function in the Y-axis direction of a given patch.
Indicators: Whether or not Close to Circumferential Part Whether or not Close to Installation Part Whether or not Close to Bend Part	close_to_circ close_to_install close_to_bend	Three binary features to indicate whether or not a given patch is close to circumferential/installation/bend parts within a distance. We set the distance as 40 cm.
Surrounding Contrast Feature	surr_contrast	For a given patch, its surrounding region is defined as a 30 cm × 30 cm area where the center is the patch itself. Surrounding contrast feature is calculated based on the same algorithm as the contrast feature discussed before.
Surrounding Patches' Individual Contrast Feature	contrast_up contrast_down contrast_left contrast_right	For a given patch, its surrounding patches are defined as 10 cm × 10 cm neighboring patches in upward, downward, leftward, and rightward directions. We calculate their contrast scores.
Surrounding Blob Area Proportion	surr_blob_prop	For a given patch, in a 30 cm x 30 cm surrounding region, we perform the blob detection (Difference of Gaussian method), which gives us several blobs (circle area). Then, we compute the proportion of blobs area to the entire surrounding region.

A.3. Case study: Pipeline magnetic flux data

Our case study is based on a pipeline section of approximately 1000 kilometers inspected using an In-Line Inspection (ILI) tool with Magnetic Flux Leakage (MFL) measurements. Automatic detection software identified 1823 candidate patches for inspection. A highly labor-intensive and authoritative human inspection process resulted in the identification of 1117 defective patches and 706 normal or non-defective patches. Fig. A.8 shows a sampling of the (a) defective and (b) normal patches. The patch sizes are 10 cm by 10 cm. Some of the normal patches are related to welds or seams in the pipeline.

The MFL path data was converted into 9 sets of features (21 individual features in total) using a combination of features from the literature and our new proposed features as shown in Table A.7 The details about how to calculate these features with MFL sensor data are presented in Appendix A.2.

The patch data was converted into 21 features using a combination of features from the literature and new features. Some of the most helpful features derived from a wavelet decomposition and sum

Table A.8
Sample of the first 10 data points and the 21 features used: (a) the first 11 features and (b) the remaining 10 features.

(a)

#	Saliency	Contrast	wavelet_H_1	wavelet_V_1	wavelet_D_1	XDP_order_1	XDP_order_2	XDP_order_3	YDP_order_1	YDP_order_2	YDP_order_3
1	0.0738	15 298.7675	79.0047	250.7065	17.0220	0.029	−0.001	0.000	−0.017	0.001	0.000
2	0.0265	12 797.3774	140.2141	521.4557	36.3180	−0.038	0.001	0.000	−0.033	0.000	0.000
3	0.0229	14 262.4727	95.4607	355.4571	24.8948	0.016	0.000	0.000	0.038	−0.001	0.000
4	0.0251	13 405.3028	85.0897	301.1001	21.2662	−0.007	0.001	0.000	0.005	0.000	0.000
5	0.0087	13 484.9795	137.8097	375.0780	27.3953	0.029	0.000	0.000	−0.011	0.000	0.000
6	0.0248	12 207.2126	89.5754	277.3459	19.1768	0.065	−0.002	0.000	−0.021	0.000	0.000
7	0.0043	13 519.4269	112.4767	460.0511	23.7697	0.032	−0.001	0.000	0.078	−0.002	0.000
8	0.0118	14 117.2736	73.4217	226.0127	17.4571	0.030	0.000	0.000	−0.028	0.000	0.000
9	0.0173	13 970.0940	110.2758	341.9894	28.1736	0.029	−0.001	0.000	−0.051	0.001	0.000
10	0.0133	13 087.5974	94.6876	344.6125	19.6914	0.028	−0.001	0.000	0.027	0.000	0.000

(b)

surr_contrast	surr_blob_prop	contrast_up	contrast_down	contrast_left	contrast_right	close_to_circ	close_to_install	close_to_bend
119 701.7325	0.0000	13 420.6738	13 971.3515	13 332.6896	13 241.8671	0	0	0
109 008.6798	0.0115	12 555.7035	12 350.7184	12 184.6820	12 221.4307	0	0	0
109 415.6816	0.0230	13 507.1450	12 709.8232	13 355.8958	12 305.3553	0	0	0
111 981.0650	0.0000	14 163.6998	13 059.8204	12 985.6345	13 670.5503	0	0	0
111 449.6536	0.0000	13 402.0357	12 222.4905	12 798.1080	12 927.3590	0	0	0
112 950.5851	0.0000	13 553.9186	12 194.3896	12 689.3747	13 101.5727	0	0	0
110 948.7129	0.0045	14 933.0134	12 616.7379	12 400.5246	12 994.8208	0	0	0
109 874.4669	0.0000	13 380.8889	13 064.5644	15 409.9544	12 273.3374	0	0	0
113 557.6528	0.0160	12 203.6066	13 037.3189	12 996.3596	12 580.1539	0	0	0
111 690.5620	0.0045	12 249.3504	12 663.9680	13 298.8263	12 047.6702	0	0	0

squaring decomposed results as suggested in previous research on hot-rolled steel parts (Ghorai et al., 2012). Also, multiple features related to contrast were derived from well-known contrast measures (Alexe et al., 2012). The resulting 21 features generated are indicated in Table A.8 parts (a) and (b). The resulting inputs were scaled to range between 0.0 and 1.0 and inputted into our Python code using the Gurobi 9.0 solver.

A.4. Pipeline results

Our results were derived using the BinNodePenalty-OCT from Section 2.3. We used a depth three tree and derived six splits. The training set included 490 normal patches and 786 defect patches. With a run time of 30 min using an i7 9th generation processor and the Gurobi 9.0 solver. The resulting tree is shown in Fig. A.9. For comparison, we also included the results based on the CART tree in Fig. A.10. The wavelet features occupy the top three nodes. Intuitively, the main split relates to how much of the feature relates to the contrasts along the diagonal. Low diagonal contrast levels likely relate to the vertical weld features indicated in Fig. A.8(b). This is further refined by vertical and horizontal contrast levels. Then contrast and salience features further divide the results. The majority of the classified images are either on the far left or the far right.

The simplicity and interpretability of the tree is a positive attribute that increases the acceptability of the model among human inspectors. The accuracy is also impressive. The model achieves 97% precision and 96% recall on the training set and 96% precision and 95% recall on the full testing set (216 normal patches and 331 defect patches). The simplicity and parsimony of the model further suggest that the results will extrapolate to some or all of the new pipes. Our team is currently exploring generalization to new pipes. In addition, the integration of the model or similar models into combined human–machine systems is being studied.

References

Aghaei, S., Azizi, M.J., Vayanos, P., 2019. Learning optimal and fair decision trees for non-discriminative decision-making. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33, (01), pp. 1418–1426.

Aghaei, S., Gomez, A., Vayanos, P., 2020. Learning optimal classification trees: Strong max-flow formulations. arXiv preprint arXiv:2002.09142.

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

Alexe, B., Deselaers, T., Ferrari, V., 2012. Measuring the objectness of image windows. IEEE Trans. Pattern Anal. Mach. Intell..

Allen, T.T., Roychowdhury, S., Liu, E., 2018. Reward-based Monte Carlo-Bayesian reinforcement learning for cyber preventive maintenance. Comput. Ind. Eng. 126, 578–594.

Bennett, K.P., Blue, J.A., 1996. Optimal decision trees. Rensselaer Polytechn. Inst. Math. Rep. 214, 24.

Bertsimas, D., Dunn, J., 2017. Optimal classification trees. Mach. Learn. 106 (7), 1039–1082.

Bertsimas, D., Dunn, J., Mundru, N., 2019. Optimal prescriptive trees. INFORMS J. Optim. 1 (2), 164–183.

Bertsimas, D., Shioda, R., 2007. Classification and regression via integer optimization. Oper. Res. 55 (2), 252–271.

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

Blanquero, R., Carrizosa, E., Molero-Río, C., Romero Morales, D., 2021. Optimal randomized classification trees. Comput. Oper. Res. 132, 105281.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.

Breiman, L., Friedman, J., Olshen, R., Stone, C., 1984. Classification and Regression Trees. Wadsworth.

Carrizosa, E., Romero Morales, D., 2013. Supervised classification and mathematical optimization. Comput. Oper. Res. 40 (1), 150–165.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining. ACM, pp. 785–794.

Firat, M., Crognier, G., Gabor, A., Hurkens, C., Zhang, Y., 2020. Column generation based heuristic for learning classification trees. Comput. Oper. Res. 116, 104866.

Ghorai, S., Mukherjee, A., Gangadaran, M., Dutta, P.K., 2012. Automatic defect detection on hot-rolled flat steel products. IEEE Trans. Instrum. Meas. 62 (3), 612–621.

Günlük, O., Kalagnanam, J., Li, M., Menickelly, M., Scheinberg, K., 2021. Optimal decision trees for categorical data via integer programming. J. Global Optim. 1–28.

Gurobi Optimization, LLC, 2020. Gurobi optimizer reference manual. http://www.gurobi.com. (Accessed 1 November 2021).

Hart, W.E., Laird, C.D., Watson, J.-P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D., 2017. Pyomo–Optimization Modeling in Python, second ed. Vol. 67, Springer Science & Business Media.

Hart, W.E., Watson, J.-P., Woodruff, D.L., 2011. Pyomo: modeling and solving mathematical programs in Python. Math. Program. Comput. 3 (3), 219–260.

ILOG CPLEX, IBM, 2020. CPLEX optimization studio 12.9.0. https://www.ibm.com/docs/en/icos/12.9.0. (Accessed 1 November 2021).

Laurent, H., Rivest, R.L., 1976. Constructing optimal binary decision trees is NP-complete. Inform. Process. Lett. 5 (1), 15–17.

Lichman, M., 2013. UCI machine learning repository. http://archive.ics.uci.edu/ml. [Retrieved March 1, 2021].

Liu, E., Allen, T.T., Roychowdhury, S., 2019. Cyber vulnerability maintenance policies that address the incomplete nature of inspection. Appl. Stoch. Models Bus. Ind. 35 (6), 1390–1410.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al., 2011. Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. 12 (Oct), 2825–2830.

Quinlan, J.R., 1986. Induction of decision trees. Mach. Learn. 1 (1), 81–106.

Quinlan, J.R., 1993. C4. 5: Programs for Machine Learning. Morgan Kaufmann, San Mateo.

Santos, L.I., Camargos, M.O., D'Angelo, M.F.S.V., Mendes, J.B., de Medeiros, E.E.C., Guimarães, A.L.S., Palhares, R.M., 2022. Decision tree and artificial immune systems for stroke prediction in imbalanced data. Expert Syst. Appl. 191, 116221.

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., Schaus, P., 2020. Learning optimal decision trees using constraint programming. Constraints 25 (3), 226–250.
Verwer, S., Zhang, Y., 2017. Learning decision trees with flexible constraints and objectives using integer optimization. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, pp. 94–103.
Verwer, S., Zhang, Y., 2019. Learning optimal classification trees using a binary linear program formulation. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33, (01), pp. 1625–1632.
Yang, L., Wang, Z., Gao, S., 2019. Pipeline magnetic flux leakage image detection algo-rithm based on multiscale SSD network. IEEE Trans. Ind. Inform. 16 (1), 501–509.