SAT-Based Approach for Learning Optimal Decision Trees with Non-Binary Features

Pouya Shati ✉
Department of Computer Science, University of Toronto, Canada

Eldan Cohen ✉
Department of Mechanical and Industrial Engineering, University of Toronto, Canada

Sheila McIlraith ✉
Department of Computer Science, University of Toronto, Canada
Vector Institute, Toronto, Canada

Abstract

Decision trees are a popular classification model in machine learning due to their interpretability and performance. Traditionally, decision-tree classifiers are constructed using greedy heuristic algorithms, however these algorithms do not provide guarantees on the quality of the resultant trees. Instead, a recent line of work has studied the use of exact optimization approaches for constructing optimal decision trees. Most of the recent approaches that employ exact optimization are designed for datasets with binary features. While numeric and categorical features can be transformed to binary features, this transformation can introduce a large number of binary features and may not be efficient in practice. In this work, we present a novel SAT-based encoding for decision trees that supports non-binary features and demonstrate how it can be used to solve two well-studied variants of the optimal decision tree problem. We perform an extensive empirical analysis that shows our approach obtains superior performance and is often an order of magnitude faster than the current state-of-the-art exact techniques on non-binary datasets.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Computing methodologies → Machine learning

Keywords and phrases Decision Tree, Classification, Numeric Data, Categorical Data, SAT, MaxSAT

Digital Object Identifier 10.4230/LIPIcs.CP.2021.50

Funding We gratefully acknowledge funding from NSERC, the CIFAR AI Chairs program (Vector Institute), and from Microsoft Research.

1 Introduction

Classification models assign class labels to data observations. Learning classification models from a set of training examples is a key task in supervised machine learning. Decision trees are among the most popular classification models in machine learning as they provide interpretable models and tend to have good performance.

Traditionally, decision-tree classifiers are constructed using greedy heuristic algorithms, such as CART [9], ID3 [21], and C4.5 [22]. However, these algorithms do not provide guarantees on the quality of the resultant trees, which can therefore be unnecessarily large or potentially inaccurate [4]. Alternatively, learning a globally optimal decision-tree classifier was shown to be NP-complete for several optimization criteria [17, 14]. Still, in recent years a variety of exact techniques have been proposed to solve the problem of optimal decision-tree classifiers. One class of techniques focuses on optimizing decision tree size (either the depth of the tree or the number of nodes in the tree) such that all training examples are correctly classified [3, 7, 18]. Another class of techniques, instead, focuses on maximizing the number of correctly classified training examples, while constraining the maximal depth of the decision tree [24, 23, 2, 15, 6]. Recent works found that optimal decision-tree classifiers tend to have

© Pouya Shati, Eldan Cohen, and Sheila McIlraith;
licensed under Creative Commons License CC-BY 4.0

higher out-of-sample accuracy than heuristic approaches [16, 3, 13, 6]. Furthermore, many of the recent approaches for optimal decision trees can be extended to support additional constraints that the resulting trees must satisfy, such as fairness constraints [1].

Many of the recent state-of-the-art approaches are designed for datasets with binary features [3, 24, 18, 15, 23], and some are further limited to binary class labels (i.e., classification with only two classes) [18, 15]. However, in practice, most real-world datasets contain categorical and numeric features. In order to perform classification in datasets with categorical and numeric features, these approaches convert any non-binary feature into a set of binary features using standard techniques (see discussions in [3, 24]). This conversion often introduces a large number of binary features and may lead to poor performance.

In this work, we present and empirically evaluate a novel approach for learning optimal decision-tree classifiers that can directly handle non-binary features without converting them into binary features. Specifically, we make the following contributions:

1. We present a novel SAT encoding of decision trees that directly supports numeric features and use this encoding to solve two well-known optimization problems in classification tasks: (1) finding a minimum-depth decision tree that correctly classifies all training examples; and (2) finding a decision tree of a given depth that maximizes the number of correctly classified training examples.
2. We present an extension of our SAT encoding that directly supports categorical features based on power set branching and show, theoretically and empirically, that the new encoding is more expressive and can lead to decision trees with better solution quality w.r.t. the two studied optimization problems.
3. We perform extensive experimental analysis and show that our encoding significantly outperforms recent state-of-the-art techniques on datasets with non-binary features for each of the studied optimization problems.

2 Technical Background

2.1 Problem Definition

In Section 2.1.1 we formally define the decision trees considered in this work and in Section 2.1.2 we define the two optimization problems we consider for learning optimal decision trees.

2.1.1 Decision Trees

We start by defining the tree structure of a decision tree. Then, we define the depth of the tree based on the deepest leaf node and the special case of complete tree.

▶ **Definition 1** (Tree Structure). *A tree structure $\mathcal{T}$ is a tuple $(\mathcal{T}_B, \mathcal{T}_L, \delta, p, l, r)$ where $\mathcal{T}_B$ and $\mathcal{T}_L$ are finite sets respectively representing the branching and leaf nodes, $\delta \in \mathcal{T}_B$ is the root node, $p : (\mathcal{T}_B \cup \mathcal{T}_L - \{\delta\}) \to \mathcal{T}_B$ is the parent function, and $l, r : \mathcal{T}_B \to (\mathcal{T}_B \cup \mathcal{T}_L)$ are respectively the left and right child functions. A well-formed tree structure is one with the property $\forall x, y : p(y) = x \leftrightarrow (l(x) = y \vee r(x) = y)$.*[1]

▶ **Definition 2** (Tree Depth). *Given a tree structure $\mathcal{T} = (\mathcal{T}_B, \mathcal{T}_L, \delta, p, l, r)$, we recursively define the depth of each node $t \in \mathcal{T}_B \cup \mathcal{T}_L$ as $depth(t) = depth(p(t)) + 1$ with $depth(\delta) = 0$. The depth of the tree structure is defined to be the maximum depth among its leaf nodes, $depth(\mathcal{T}) = \max_{t \in \mathcal{T}_L} depth(t)$.*

[1] Note that well-formedness guarantees the absence of loops in parental relations.

▶ **Definition 3** (Complete Tree). *A tree structure $\mathcal{T} = (\mathcal{T}_B, \mathcal{T}_L, \delta, p, l, r)$ is considered complete if all the leaf nodes have the same depth, i.e., $\forall t_1, t_2 \in \mathcal{T}_L : depth(t_1) = depth(t_2)$.*

Next, we formally define decision trees and describe how to perform prediction using such decision trees.

▶ **Definition 4** (Decision Tree). *Given a set of features F and integer labels C, a decision tree is a tuple $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ where $\mathcal{T} = (\mathcal{T}_B, \mathcal{T}_L, \delta, p, l, r)$ is a tree structure, $\beta : \mathcal{T}_B \to F$ is the feature selection function, $\alpha : \mathcal{T}_B \to dom(F)$ is the threshold selection function, and $\theta : \mathcal{T}_L \to C$ is the leaf labelling function. A well-formed decision tree satisfies $\forall t \in \mathcal{T}_B : \alpha(t) \in dom(\beta(t))$. Note that $dom(j)$ represents the set of possible values for feature $j \in F$ and $dom(F) = \bigcup_{j \in F} dom(j)$.*

An evaluation of a set of features F is called a data point x. For each $j \in F$, $x[j] \in dom(j)$ represents the value of feature j at point x. A dataset usually referred to as X, contains a finite set of data points $x_i \in X$ for training or testing purposes.

A decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ predicts the label of a given point x_i by starting from the root and recursively delivering the point to the left or right child until a leaf node is reached. The label of the leaf node is the output. The recursive *predict* function $\Theta(t, x_i)$ on node $t \in \mathcal{T}_B \cup \mathcal{T}_L$ and point $x_i \in X$ is defined as follows:

$$\Theta(t, x_i) = \begin{cases} \theta(t) & \text{if } t \in \mathcal{T}_L \\ \Theta(l(t), x_i) & \text{elseif } x_i[\beta(t)] \leq \alpha(t) \\ \Theta(r(t), x_i) & \text{else} \end{cases}$$

The prediction of decision tree $\mathcal{D}$ for data point $x' \in X$ can be obtained by $\Theta(\delta, x')$ where δ is the root node of the decision tree. We use the simplified notation $\Theta(x')$ to denote $\Theta(\delta, x')$.

Given a labelled dataset, i.e., a dataset in which the correct class label for each data point is provided, we can measure the accuracy of a decision tree on the dataset with respect to the provided labels.

▶ **Definition 5** (Decision Tree Accuracy). *Given a set of features F and integer labels C, a set of training examples $x_i \in X$, and a labelling $\gamma : X \to C$, the accuracy of decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ on X is the fraction of data points in X that are correctly classified with respect to the labelling γ,*

$$\frac{\sum_{x_i \in X} \mathbb{1}[\Theta(x_i) = \gamma(x_i)]}{|X|},$$

where $\mathbb{1}$ is the indicator function.

2.1.2 Learning Optimal Decision Trees

We consider two problems representing two different optimization criteria for optimal decision tree. Problem 1 consists of finding a decision tree with minimum depth such that all the training examples are correctly classified. This problem is consistent with Avellaneda [3].

▶ **Problem 1** (Min-Depth Optimal Decision Tree). *Given set of features F and integer labels C, a set of training examples $x_i \in X$, and a labelling $\gamma : X \to C$, output a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ such that $\mathcal{T}$ is a minimum depth complete tree and $\mathcal{D}$ correctly classifies all training examples, i.e., $\Theta(x_i) = \gamma(x_i)\ \forall x_i \in X$.*

Problem 2 consists of finding a decision tree that maximizes the number of training examples that are correctly classified subject to a constraint on the tree depth. This problem is consistent with the problems considered in a variety of recent works, e.g., in [15, 23, 2].

▶ **Problem 2** (Max-Accuracy Optimal Decision Tree)**.** *Given a set of features F and integer labels C, a set of training examples $x_i \in X$, a labelling $\gamma : X \to C$, and a chosen depth d, output a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ such that $\mathcal{T}$ is a complete tree of the chosen depth, $depth(\mathcal{T}) = d$, and $\mathcal{D}$ maximizes the number of training examples that are correctly classified, i.e., $\sum_{x_i \in X} \mathbb{1}[\Theta(x_i) = \gamma(x_i)]$ where $\mathbb{1}$ is the indicator function.*[2]

2.2 SAT and MaxSAT

SAT formulae are represented in Conjunctive Normal Form (CNF) and are defined over a set of Boolean variables. A SAT formula is a conjunction of clauses, each clause is a disjunction of literals, and each literal is either a Boolean variable or its negation. An assignment of the Boolean variables satisfies a clause if at least one of its literals is true. The SAT problem consists of finding an assignment of the variables that satisfies all clauses in a formula [8].

The MaxSAT problem is the optimization variant of the SAT problem and consists of finding an assignment of the variables that maximizes the number of satisfied clauses. Partial MaxSAT [12] is a generalization of the MaxSAT problem where the set of clauses consists of *hard clauses* that must be satisfied and *soft clauses* that can be violated.

3 SAT-based Encoding for Learning Optimal Decision Trees

In this section, we present our SAT-based approach for optimal decision trees. In Section 3.1, we present the core encoding for decision trees that can be used to solve the two optimization problems considered in this work. In Section 3.2, we present a SAT-based approach for solving the min-depth problem (Problem 1) by searching for increasingly deeper decision trees that can correctly classify all training examples. In Section 3.3, we present a partial MaxSAT encoding of the max-accuracy problem (Problem 2).

3.1 Encoding Decision Trees

We propose a SAT encoding of a decision tree, $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$. Similar to previous works (e.g., [3, 23]), our encoding assumes a tree structure $\mathcal{T}$ (typically a complete tree of some depth d), and decides on the values of β, α, and θ. When the depth of the tree is unknown in advance (e.g., in the min-depth optimal decision tree problem) we can solve a sequence of problems for trees of increasing depth as described in Section 3.2.

3.1.1 Variables

The following binary variables are used to represent the different aspects of a decision tree:

- $[a_{t,j}]$: Represents whether feature j is chosen for the split at branching node t.
- $[s_{i,t}]$: Represents whether point i is directed towards the left child, if it passes through branching node t.
- $[z_{i,t}]$: Represents whether point i ends up at leaf node t.
- $[g_{t,c}]$: Represents whether label c is assigned to leaf node t.

[2] Note that since $|X|$ is fixed, maximizing the number of correctly classified training examples is identical to maximizing the accuracy in Definition (5).

3.1.2 Clauses

The following set of hard clauses in conjunctive normal form guarantee the validity of the recursion in the modelled decision tree, and can consequently be used as the core encoding for both of the optimization problems we consider.

The clauses in Eq. (1) and Eq. (2) guarantee that exactly one feature is chosen at each branching node $t \in \mathcal{T}_B$.

$$(\neg a_{t,j}, \neg a_{t,j'}) \qquad t \in \mathcal{T}_B, j \neq j' \in F \tag{1}$$

$$(\bigvee_{j \in F} a_{t,j}) \qquad t \in \mathcal{T}_B \tag{2}$$

For each branching node $t \in \mathcal{T}_B$, we need to make sure that all the data points for which the feature value is less than or equal to the feature threshold are directed left and all the data points for which the feature value is greater than the threshold are directed right. We use $\#^i_j$ to denote the index of the i'th data point in X when sorted by feature j in ascending order, assuming ties are broken arbitrarily, and define O_j to be the set of all consecutive pairs in this ordering, i.e., $O_j = \{(\#^i_j, \#^{i+1}_j) \mid 1 \leq i \leq |X|-1\}$. Then, the clauses in Eq. (3) guarantee that there are no two points with different feature values where the one with the higher value is directed left while the one with the lower value is directed right. The clauses in Eq. (4), together with the clauses in Eq. (3), guarantee that points with equal values are directed in a similar manner.

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i, i') \in O_j(X) \tag{3}$$

$$(\neg a_{t,j}, \neg s_{i,t}, s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i, i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{4}$$

For each data point x_i, we need to guarantee the validity of its path in the decision tree. We use $A_l(t)$ (resp. $A_r(t)$) to denote all the ancestors of a leaf node $t \in \mathcal{T}_L$ such that t is a descendant of their left (resp. right) branch. The clauses in Eq. (5) and Eq. (6) guarantee that each data point that ends up at a leaf node follows the corresponding path. In contrast, the clauses in Eq. (7) guarantee that each data point that does not end up in leaf node $t \in \mathcal{T}_L$ has at least one deviation from the corresponding path

$$(\neg z_{i,t}, s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_l(t) \tag{5}$$

$$(\neg z_{i,t}, \neg s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_r(t) \tag{6}$$

$$(z_{i,t}, \bigvee_{t' \in A_l(t)} \neg s_{i,t'}, \bigvee_{t' \in A_r(t)} s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{7}$$

The clauses in Eq. (8) guarantee that each leaf node is assigned at most one label. Note that we do not include constraints that prevent leaves with no label. The optimization criteria discussed in Sections 3.2 and 3.3 guarantee that in optimal solutions, all leaves that have corresponding training examples will be assigned a label. Leaf nodes that do not have any corresponding training examples will be assigned an arbitrary label post-optimization in order to maintain a valid decision tree.

$$(\neg g_{t,c}, \neg g_{t,c'}) \qquad t \in \mathcal{T}_L, c \neq c' \in C \tag{8}$$

Finally, we add redundant constraints that help prune the search space. For each branching node, the clauses in Eq. (9) guarantee that the data point with the lowest feature value is directed left and the clauses in Eq. (10) guarantee that the data point with the highest feature value is directed right.

$$(\neg a_{t,j}, s_{\#^1_j,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{9}$$

$$(\neg a_{t,j}, \neg s_{\#^{|X|}_j,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{10}$$

3.1.3 Decoding Decision Trees from Solutions

Assuming a solution to the SAT encoding above, i.e., an assignment of the variables $a_{t,j}$, $s_{i,t}$, $z_{i,t}$, and $g_{t,c}$ that satisfy the clauses in Section 3.1.2, we now describe how to extract the decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$. Decoding β is done by setting $\beta(t) = j$ if the variable $a_{t,j}$ is true. Similarly, decoding θ is done by setting $\theta(t) = c$ if the variable $g_{t,c}$ is true. Note that these procedures are valid since we are guaranteed that $a_{t,j}$ and $g_{t,c}$ are unique for each node, i.e., $\forall t \in \mathcal{T}_B : \sum_{j \in F} a_{t,j} = 1$ and $\forall t \in \mathcal{T}_L : \sum_{c \in C} g_{t,c} = 1$.

Since our SAT encoding does not explicitly compute the threshold for each node, to decode α we have to choose a threshold based on the direction of the data points in each branching node. For a branching node $t \in \mathcal{T}_B$ with $\beta(t) = j$, we set $\alpha(t) = x_i[j]$ where $(i, i') \in O_j(X)$ are consecutive data points according to the ordering of feature j such that $s_{i,t}$ is true and $s_{i',t}$ is false. Intuitively, this rule uses the largest value directed left as the feature threshold for the node t.

3.2 Encoding the Min-Depth Optimal Decision Tree Problem

To find a minimum depth decision tree such that all training examples are correctly classified, we add the following clauses to the core decision tree encoding described above:

$$(\neg z_{i,t}, g_{t,\gamma(x_i)}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{11}$$

The clauses in Eq. (11) guarantee that the class labels assigned to leaf nodes are consistent with the training set labels of the corresponding data points. If a data point x_i ends in leaf node $t \in \mathcal{T}_L$ then the assigned label of t must match the training label $\gamma(x_i)$.

In order to guarantee that we find the minimum depth decision tree, we follow the technique in [3]. We start by solving the SAT formula for a tree structure $\mathcal{T}$ of depth 1 and in each iteration increase the depth by 1 until a solution is found. This guarantees that when the SAT solver finds a solution, the obtained decision tree has a minimum depth subject to the constraint that all training examples must be classified correctly.

3.3 Encoding the Max-Accuracy Optimal Decision Tree Problem

To find a decision tree of a given depth that maximizes the number of correctly classified training examples, we introduce the following variables to keep track of training examples that are correctly classified:

- $[p_i]$: Represents whether point x_i is correctly classified, i.e., x_i ends up in a leaf node with the label $\gamma(x_i)$.

We use a partial MaxSAT model that includes both hard and soft clauses. We use all the clauses from our core decision tree encoding in Section 3.1 as hard clauses. We also add the hard clauses in Eq. (12) that guarantee p_i is set to true only when x_i ends up in a leaf node whose label is consistent with the training label.

$$(\neg p_i, \neg z_{i,t}, g_{t,\gamma(x_i)}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{12}$$

In order to maximize the number of correctly classified training examples, we add a soft clause for each data point, that is satisfied whenever the point is correctly classified.

$$(p_i) \qquad x_i \in X \tag{13}$$

Therefore, the optimal solution for this partial MaxSAT model is a decision tree that maximizes the number of correctly classified training examples.

4 Extending Optimal Decision Trees for Categorical Features

Categorical features are features that take their value from a set of values representing different categories that do not induce a natural ordering.[3] For example, in a dataset of financial transactions we can have a feature that describes the type of transaction from a set of known transaction types. In order to deal with a categorical feature with K categories, one can convert the feature to K binary features representing a one-hot encoding of the original categorical feature. However, this can lead to unnecessarily deep decision trees as splits may be required for many categories.

In this section, we show that we can easily extend our approach to generate decision trees that support branching on categorical features directly. We employ *power set branching* in which a selected subset of the categories is assigned to the left branch while the subset that contains the rest of the categories is assigned to the right branch. For the min-depth optimal decision tree problem, such extension can lead to decision trees of smaller depth that can potentially be found earlier. For the max-accuracy optimal decision tree problem, such extension can allow more flexible trees that achieve higher accuracy for the same depth.

▶ **Example 1.** *To demonstrate the potential benefit of power set branching, consider a simplified dataset of transactions that can either be approved or denied. Each transaction has one categorical feature describing the transaction type with the categories $\{A, B, C, D, E, F\}$. Transaction with types A, D, and F are approved, while transactions with types B, C, and E are denied. Figure 1 (left) shows a standard decision tree where the categorical feature was converted to six binary features representing a one-hot encoding of the original feature. Figure 1 (right) shows the extended variant of decision trees where we can branch directly on categorical features by assigning a subset of the categories to each branch. The standard decision tree requires branching, in sequence, on multiple different categories leading to a tree with a minimum depth of 3. In contrast, the extended decision tree that supports branching on categorical features has a depth of 1.*

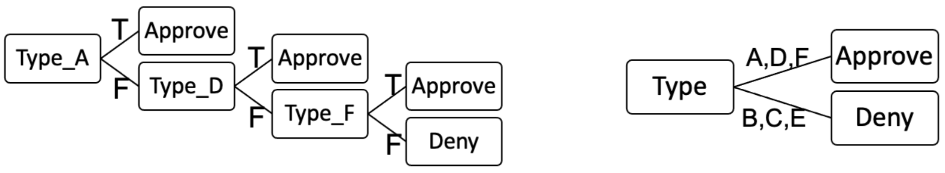

Figure 1 Left: a standard decision tree for the example dataset. Right: a decision tree with power set branching for categorical features.

4.1 Decision Trees with Power Set Branching for Categorical Features

We assume a set of features F that includes categorical features F_C and numeric features F_N such that $F = F_C \cup F_N$. In order to support power set branching, we augment the decision tree with a category subset selector α_C, $\mathcal{D}_C = (\mathcal{T}, \beta, \alpha, \alpha_C, \theta)$. We denote the set of branching nodes associated with numeric features $\Pi^N = \{t \in \mathcal{T}_B \mid \beta(t) \in F_N\}$ and similarly define Π^C the set of branching nodes associated with categorical features. We use α as a

[3] Some categorical features induce a natural ordering and can therefore be represented as numeric features. For example a categorical feature with the categories {Low, Medium, High} can be transformed to a numeric feature with the values {1, 2, 3}.

threshold selector for branching nodes with numeric features, $\alpha : \Pi^N \to dom(F_N)$, and α_C that selects a subset of categories for branching nodes with categorical feature from the corresponding power set $\alpha_C : \Pi^C \to 2^{dom(F_C)}$.[4]

Intuitively, α_C provides the subset of categories for which data points should be directed left. To predict the label of a given point x_i we use the recursive Θ_C as follows:

$$\Theta_C(t, x_i) = \begin{cases} \theta(t) & \text{if } t \in \mathcal{T}_L \\ \Theta_C(l(t), x_i) & \text{elseif } (t \in \Pi^N \wedge x_i[\beta(t)] \leq \alpha(t)) \vee (t \in \Pi^C \wedge x_i[\beta(t)] \in \alpha_C(t)) \\ \Theta_C(r(t), x_i) & \text{else} \end{cases}$$

Decision trees with power set branching are more expressive than decision trees that use a binarized encoding of these categorical features. Specificially, each decision tree that uses binary branching for categorical features can be transformed into a decision tree with power set branching with identical predictions.

▶ **Proposition 1.** *Given a decision tree $\mathcal{D} = (\mathcal{T}, \beta, \alpha, \theta)$ operating on a set of features $bin(F)$, there exists a decision tree with power set branching on the same structure $\mathcal{D}'_C = (\mathcal{T}, \beta', \alpha, \alpha'_C, \theta)$ operating on the set of features F such that*

$$\forall x_i : \Theta_C(x_i) = \Theta(bin(x_i))$$

where $bin(F)$ is F with its categorical features encoded using binary features in one-hot style and $bin(x_i)$ is x_i with its values encoded according to the binarized features set $bin(F)$.

The above proposition is correct since each branching node in the decision tree $\mathcal{D}$ associated with a binary feature that correspond to one category c in the categorical feature $j \in F_C$ can be replaced in $\mathcal{D}'$ with a node that branches directly on the feature j and directs the subset of categories $\{c\}$ to the right and the subset that contains the rest of the categories to the left. Proposition 1 implies the following corollaries on the solution-quality guarantees of power set branching w.r.t each of the optimization problems.

▶ **Corollary 6.** *Given an instance of Problem 1, the minimum depth found for a decision tree with power set branching is always equal to or less than that of a decision tree with branching based on binarized encoding of categorical features.*

▶ **Corollary 7.** *Given an instance of Problem 2, the maximum accuracy found for a decision tree with power set branching is always equal to or more than that of a decision tree with branching based on binarized encoding of categorical features.*

4.2 SAT-based Encoding of Decision Trees with Power Set Branching

Interestingly, the proposed extension involves only minor changes to the decision tree encoding in Section 3.1. Eq. (14)–(24) show the modified encoding with the changes highlighted in blue. The clauses in Eq. (16) guarantee that data points where the feature value is less or equal to the feature threshold are directed left and vice versa. As this does not apply to categorical features, we restrict Eq. (16) to numeric features. Instead, we add Eq. (18) that, together with the existing Eq. (17), guarantees that data points with the same category are directed in the same direction. Finally, for numeric features we used redundant constraints

[4] Similar to α, a well-formedness condition on α_C would dictate that $\forall t \in \Pi^C : \alpha_C(t) \subseteq dom(\beta(t))$.

that guarantee the lowest feature value is directed left and the highest feature value is directed right. For categorical features we do not have such ordering over categories and instead we simply use the clauses in Eq. (23) to make sure that some arbitrary category is chosen to be directed left (in categorical features, $\#_j$ represents an arbitrary ordering over the category values of feature j) and we modify Eq. (24) such that no specific category is forced to go right.[5] Note that unlike previous work that focused on categorical features [13], our approach directly encodes optimal decision trees with both numeric and categorical features.

$$(\neg a_{t,j}, \neg a_{t,j'}) \qquad t \in \mathcal{T}_B, j \neq j' \in F \tag{14}$$

$$(\bigvee_{j \in F} a_{t,j}) \qquad t \in \mathcal{T}_B \tag{15}$$

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F_N, (i,i') \in O_j(X) \tag{16}$$

$$(\neg a_{t,j}, \neg s_{i,t}, s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F, (i,i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{17}$$

$$(\neg a_{t,j}, s_{i,t}, \neg s_{i',t}) \qquad t \in \mathcal{T}_B, j \in F_C, (i,i') \in O_j(X), x_i[j] = x_{i'}[j] \tag{18}$$

$$(\neg z_{i,t}, s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_l(t) \tag{19}$$

$$(\neg z_{i,t}, \neg s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X, t' \in A_r(t) \tag{20}$$

$$(z_{i,t}, \bigvee_{t' \in A_l(t)} \neg s_{i,t'}, \bigvee_{t' \in A_r(t)} s_{i,t'}) \qquad t \in \mathcal{T}_L, x_i \in X \tag{21}$$

$$(\neg g_{t,c}, \neg g_{t,c'}) \qquad t \in \mathcal{T}_L, c \neq c' \in C \tag{22}$$

$$(\neg a_{t,j}, s_{\#_j^1,t}) \qquad t \in \mathcal{T}_B, j \in F \tag{23}$$

$$(\neg a_{t,j}, \neg s_{\#_j^{|X|},t}) \qquad t \in \mathcal{T}_B, j \in F_N \tag{24}$$

Note that in our encoding, it is possible to have degenerate nodes for which all categories are directed left with none of the categories directed right. An optimal solution may have degenerate nodes, however we can easily convert such solutions to optimal solutions without degenerated nodes. As a post-optimization step, we arbitrarily select a subset of categories from the left branch and move them to the right branch. Then, we copy all the subtree of the left branch to the right branch, leading to identical predictions on the training examples and maintaining the optimality of the original solution.

4.2.1 Decoding Decision Trees with Power Set Branching

Decoding a decision tree from a solution to the SAT encoding above follows the same procedure as described in Section 3.1.3, with one key difference. For nodes that are associated with categorical branching, we need to decode α_C, the category subset selector that indicates the subset of categories associated with the left branch. For a categorical branching node $t \in \Pi^C$ that is associated with feature j, i.e., $\beta(t) = j, j \in F_C$, we define $\alpha_C(t)$ to be the set of categories in feature j for which there are data points in X that are directed left,

$$\alpha_C(t) = \{c \mid \exists x_i \in X : x_i[\beta(t)] = c \wedge s_{i,t} = 1\}.$$

[5] Note that we could add clauses that guarantee that at least one category will go to the right, however it does not provide significant pruning and we therefore opted not to add them.

5 Experiments

In this section, we perform an extensive experimental evaluation of our SAT-based approach for optimal decision trees. For each of the two optimization criteria, we compare our approach to state-of-the-art exact approaches for optimal decision trees that optimize the same criterion based on the runtime and objective value. Previous work on each of the two criteria has already evaluated the quality of decision trees that are optimal with respect to a set of training examples on external validation datasets [15, 3, 2, 24] and also discussed the differences between the two criteria [15, 3]. In this work, we focus on the optimization performance with respect to each of the optimization criteria.

Our algorithm for the min-depth optimal decision tree problem is implemented in C++ based on SAT solver MiniSAT [11]. Our encoding for the max-accuracy optimal decision tree problem is solved using the MaxSAT solver Loandra [5]. The experiments were run on two 12-core Intel E5-2697v2 CPUs and 128G of ram.

5.1 Baselines

We compare our approach to recent state-of-the-art baselines for each optimization problem. For min-depth optimal decision trees, we compare our approach to Avellaneda's SAT-based approach [3] that uses the MiniSAT solver [11]. For max-accuracy optimal decision trees, we compare our approach to Hu et al.'s MaxSAT encoding [15] solved by Loandra [5], and to Verhaeghe et al.'s constraint programming (CP) model [23] solved by Oscar [20]. Note that other approaches, such as OCT [6], BinOCT [24], DL8 [19], DL8.5 [2], have been previously compared to one or more of the baselines we consider in their original publications [15, 3, 23].

Binary Encoding of Non-Binary Features

Most recent state-of-the-art approaches, including the selected baselines, support datasets that contain only binary features. Therefore, these approaches convert any non-binary feature into a set of binary features in a pre-processing step prior to optimization. To binarize the datasets, we follow the procedure in Avellaneda [3] in which each non-binary feature that can have v possible different values is represented by v binary features, one for each possible value. Furthermore, if the feature is ordered, e.g., numeric features, then each binary feature represents the operator $\leq$ as described in the following example from Avellaneda [3]:

▶ **Example 2.** *Consider a set of training examples where each example has a single integer feature f, {(1), (3), (4), (5)}. Then, we can transform f into three binary features $\tilde{f}_0, \tilde{f}_1, \tilde{f}_2$. If the feature $\tilde{f}_0$ is true it means that the value of f in the example is greater than* 1*. If the feature $\tilde{f}_1$ is true it means that the value of f in the example is greater than* 3*, etc. Therefore, the transformed dataset will be {(0,0,0), (1,0,0), (1,1,0), (1,1,1)}.*

Note that the above binary feature encoding supports the decision trees described in Definition 4 since selecting a binary feature for a branching node implies a threshold that values smaller or equal to the threshold are directed into one branch, while values greater than the threshold are directed into the other branch.

5.2 Datasets

To evaluate the performance of our approach,we run experiments on 15 datasets with different characteristics, obtained from the UCI repository [10]. Table 1 reports the size of each dataset $|X|$, the number of class labels $|C|$, the number of numeric features $|F_N|$, the number of

Table 1 Description of the datasets used in our experiments.

Type	Name	$\|X\|$	$\|F_N\|$	$\|F_B\|$	$\|F_C\|$	$\tilde{f}$	$\|C\|$
N	Banknote	1372	4	0	0	5016	2
	Breast Cancer	116	9	0	0	891	2
	Cryotherapy	90	5	1	0	93	2
	Immunotherapy	91	6	1	0	166	2
	Ionosphere	351	32	2	0	8114	2
	Iris	150	4	0	0	119	3
	User Knowledge	258	5	0	0	431	4
	Vertebral Column	310	6	0	0	1741	2
	Wine	178	13	0	0	1263	3
C	Credit Approval†	653	6	4	5	1130	2
	Promoter	106	0	0	57	228	2
	Soybean Large†	266	5	16	14	76	15
	Protease Cleavage	746	0	0	8	160	2
	Protease Cleavage(/4)	186	0	0	8	156	2
B	Car‡	1728	6	0	0	15	2
	Monk2	169	4	2	0	11	2

† Records with missing values were removed.
‡ Classes were merged following Avellaneda [3].

binary features[6] $|F_B|$, and the number of categorical features $|F_C|$. It also reports the number of binary features after the transformation of all the numeric and categorical features into binary features which is required for the baseline methods.

Consistent with our focus on non-binary features, we selected datasets with mostly numeric features (type N) or categorical features (type C). We also included the datasets Monk2 and Car that we consider binary (type B) since they have either binary features or numeric features that can be convert to binary with a small number of additional variables.

The datasets Credit Approval, Promoter, Soybean Large, and Protease Cleavage include a significant number of categorical features and will be used to evaluate our extension for optimal decision trees with categorical features. We also created a smaller version of Protease Cleavage that includes only 25% of the records (by selecting each fourth row).

5.3 Results

We start by evaluating our SAT-based formulation in Section 3 on numerical and binary datasets. In Section 5.4, we present the results for the min-depth optimal decision tree problem and in Section 5.5, we present the results for the max-accuracy optimal decision tree problem. In Section 5.6, we present results for the categorical branching extension in Section 4. Finally, in Section 5.7 we analyze the memory consumption.

[6] In our encoding binary features are numeric features that take one of two possible values, however we list them separately in Table 1 as they are supported by the baseline methods without transformation.

Table 2 Experimental results for min-depth optimal decision tree problem.

Dataset	Min Depth	Time (s)	
		Ours	SAT [3]
Banknote	4	**5.82**	T/O [4]
Breast Cancer	4	**6.59**	T/O [4]
Cryotherapy	4	**0.08**	0.24
Immunotherapy	4	**0.18**	1.3
Ionosphere	?	T/O [4]	T/O [3]
Iris	4	**0.04**	0.17
User Knowledge	5	**1.31**	59.44
Vertebral Column	5	**87.35**	T/O [5]
Wine	3	**0.11**	14.75
Car	8	T/O [8]	**89.1**
Monk2	6	2.73	**0.28**

5.4 Results for the Min-Depth Optimal Decision Tree Problem

We compare our encoding for min-depth optimal decision trees to Avellaneda's SAT encoding [3] on the binary and numeric datasets. Following Avellaneda's analysis, we set the time limit to 30 minutes. Table 2 shows the time to optimal solution for each of the approaches, as well as the tree depth of the optimal solution. As both approaches follow the procedure of solving SAT formulae for increasingly deeper trees until a solution is found, the first solution found is guaranteed to be optimal, i.e., of minimum depth. In case an approach does not find an optimal solution in the time limit, we report "T/O [d]" where d indicates the tree depth of the SAT formula being solved at the moment of time out. We highlight in bold results for which the alternative approach required at least twice as much run time or timed out.

The results in Table 2 show that our approach performs at least as well, and in most cases significantly better on datasets with numeric features. In particular, it finds optimal solutions for Banknote, Breast Cancer and Vertebral Column, for which the baseline timed out. In Ionosphere we find that both methods timed out, however our approach has managed to prove that a solution does not exist for a depth of 3, while the baseline only proved that a solution does not exist for a depth of 2. As expected, in the two binary datasets (Car and Monk2), we find that the baseline outperforms our method. In particular, it manages to find an optimal solution to Car, while our approach timed out.

5.5 Results for the Max-Accuracy Optimal Decision Tree Problem

Next, we compare our encoding for max-accuracy optimal decision trees to Hu et al.'s MaxSAT encoding [15] and Verhaeghe et al.'s CP encoding [23] on the binary and numeric datasets. Following Hu et al., we set the time limit to 15 minutes and run experiments for three different depth values, $\{2, 3, 4\}$. Table 3 shows the time required to find an optimal solution for each approach or "T/O" if an optimal solution was not found in the time limit. It also reports the cost of the solutions, i.e., the number of training examples that are *not* correctly classified, for each approach. If an optimal solution was found and proved, then the cost indicates the optimal cost. Otherwise, we report the cost of the best found solution by each approach. Note that the datasets Iris, User Knowledge, and Wine could only be solved by our approach as the two baselines only support classification problems with two class labels.

Table 3 Experimental results for max-accuracy optimal decision tree problem.

Dataset	Depth	Solution Cost			Time (s)		
		Ours	MaxSAT [15]	CP [23]	Ours	MaxSAT [15]	CP [23]
Banknote	2	**100**	176	**100**	**16.83**	T/O	512.21
	3	**23**	550	100	**105.79**	T/O	T/O
	4	**0**	88	100	**18.98**	T/O	T/O
Breast Cancer	2	**19**	24	**19**	**5.07**	T/O	22.19
	3	**9**	25	12	**242.16**	T/O	T/O
	4	**0**	18	11	**20.79**	T/O	T/O
Cryotherapy	2	**5**	**5**	**5**	**0.57**	3.68	4.21
	3	**1**	**1**	**1**	**0.73**	17.57	27.39
	4	**0**	**0**	**0**	**0.75**	24.14	7.61
Immunotherapy	2	**8**	**8**	**8**	**0.99**	10.53	5.22
	3	**4**	**4**	**4**	**3.81**	T/O	146.45
	4	**0**	1	**0**	**1.27**	T/O	18.53
Ionosphere	2	**29**	41	**29**	**155.06**	T/O	T/O
	3	**21**	186	29	T/O	T/O	T/O
	4	**10**	76	28	T/O	T/O	T/O
Iris	2	**6**	–	–	**0.6**	–	–
	3	**1**	–	–	**0.77**	–	–
	4	**0**	–	–	**0.82**	–	–
User Knowledge	2	**35**	–	–	**1.94**	–	–
	3	**10**	–	–	**3.29**	–	–
	4	**1**	–	–	**3.86**	–	–
Vertebral Column	2	**45**	46	**45**	**15.79**	T/O	67.91
	3	**32**	44	42	T/O	T/O	T/O
	4	**15**	39	42	T/O	T/O	T/O
Wine	2	**6**	–	–	**1.25**	–	–
	3	**0**	–	–	**1.62**	–	–
Car	2	**250**	**250**	**250**	12.67	9.2	**2.16**
	3	**182**	**182**	**182**	T/O	T/O	**5.99**
	4	**122**	**122**	**122**	T/O	T/O	**14.09**
Monk2	2	**57**	**57**	**57**	2.74	4.38	1.38
	3	**42**	**42**	**42**	T/O	826.31	**3.6**
	4	32	**31**	**31**	T/O	T/O	**8.12**

The results in Table 3 show that for all numeric datasets our approach either finds solutions faster than the baselines or that all approaches time out. Furthermore, for all numeric datasets, our approach finds solutions that are at least equal, and in many cases significantly better than the baselines. As expected, for binary datasets (Car and Monk2), our approach underperforms relative to the baselines and timed out for depths 3 and 4. Still, it finds the optimal solution for all depths in Car and for depth 2 and 3 in Monk2. For the three datasets that could not be solved by the baselines, our approach found optimal solutions in seconds. In Wine, a tree of depth 3 correctly classifies all training examples.

Table 4 Results for min-depth decision trees on datasets with categorical features.

Dataset	Min Depth			Time (s)		
	Ours-PS	Ours	SAT [3]	Ours-PS	Ours	SAT [3]
Credit Approval	?	?	?	T/O [6]	T/O [5]	T/O [5]
Protease Cleavage	?	?	?	T/O [5]	T/O [7]	T/O [6]
Protease Cleavage(/4)	**4**	?	?	0.69	T/O [7]	T/O [6]
Promoter	**4**	**4**	**4**	224.22	498.3	87.69
Soybean Large	?	?	?	T/O [6]	T/O [6]	T/O [6]

5.6 Results on Categorical Datasets

In this section, we present results on datasets with categorical features. We compare our power set branching for categorical features (Ours-PS) against our encoding without power set branching (Ours) and the baselines. As the optimal solution for the different approaches may be different, we do not highlight in bold the lowest run time. For example, our power set branching can fail to find an optimal solution in the time limit, while still obtaining a lower-cost solution compared to an optimal solution of a binary branching approach.

Table 4 reports the results for the min-depth optimal decision tree problem. Note that we report the min depth for each of the approaches as the power set approach can have a lower optimal solution. We find that most categorical datasets could not be solved by any of the methods in the time limit. However, in Protease Cleavage(/4), our encoding that is based on power set branching (Ours-PS) is able to find a solution decision tree of depth 4, while binary branching fails to find a solution after proving that no solution exists for a depth of 6. This demonstrates the expressiveness of our power set branching for categorical features.

Table 5 shows the results for the max-accuracy optimal decision tree problem. We find that in all of the cases, our power set approach obtains the lowest cost solution and that in almost all cases, these solutions are strictly lower compared to all other approaches. Note that we encountered an error when running the code for the CP approach [23] on the Credit Approval dataset with a depth of 4.

5.7 Results on Memory Consumption

To compare the memory consumption of the different approaches, we recorded the peak memory consumption of each approach for each of the datasets. Due to limited space, we only report aggregated results. Table 6 reports the mean and maximum values for the different approaches for each optimization problem and dataset type. In all cases, our approach has the lowest mean and maximum values. In categorical datasets, our power set encoding obtains the second lowest values. In the worst case, our approach required approximately 1GB of memory for the standard encoding and 1.9GB for the power set encoding. In comparison, Avellaneda [3] and Hu et al. [15] required more than 10GB in the worst case. For Verhaeghe et al. [23], we find that the maximum values are approximately within a factor of two from ours, however the mean values are still significantly higher than ours.

6 Conclusion

We present a novel SAT-based encoding for optimal decision trees that can directly encode non-binary features, namely numeric and categorical features. We study two variants of optimal decision trees based on different optimization criteria and present extensive empirical

Table 5 Results for max-accuracy decision trees on datasets with categorical features.

Dataset	Depth	Cost				Time (s)			
		Ours-PS	Ours	[15]	[23]	Ours-PS	Ours	[15]	[23]
Credit Approval	2	**84**	**84**	**84**	**84**	106.26	151.45	T/O	33.06
	3	**76**	**76**	82	81	T/O	T/O	T/O	T/O
	4	**60**	65	74	[E]	T/O	T/O	T/O	[E]
Protease Cleavage	2	**98**	186	186	186	T/O	325.35	56.11	4.82
	3	**101**	133	149	133	T/O	T/O	T/O	29.68
	4	**39**	98	136	100	T/O	T/O	T/O	T/O
Protease Cleavage(/4)	2	**13**	49	49	49	18.91	28.58	16.5	4.36
	3	**1**	29	29	29	7.37	575.93	T/O	22.7
	4	**0**	37	40	17	1.27	T/O	T/O	T/O
Promoter	2	**12**	13	13	13	316.71	44.53	316.02	4.35
	3	**3**	**3**	4	**3**	T/O	324.77	T/O	63.22
	4	**0**	**0**	**0**	**0**	7.83	57.72	81.08	129.15
Soybean Large	2	**152**	159	–	–	16.22	99.93	–	–
	3	**104**	112	–	–	T/O	T/O	–	–
	4	**35**	45	–	–	T/O	T/O	–	–

Table 6 Analysis of peak memory consumption (MB) for the different approaches.

		Min-Depth			Max-Accuracy			
		Ours-PS	Ours	[3]	Ours-PS	Ours	[15]	[23]
N	Mean	N/A	**142.02**	4,527.80	N/A	**236.34**	1,646.73	1,176.36
	Max	N/A	**1,003.86**	16,684.36	N/A	**1,299.09**	10,994.48	2,381.50
C	Mean	904.73	**489.42**	3,086.98	412.67	**222.61**	728.83	1,332.50
	Max	1,865.21	**984.20**	10,075.11	1,451.05	**705.93**	3,838.78	2,180.96

analysis that shows our approach outperforms recent state-of-the-art methods on datasets with non-binary features in terms of optimization quality. Furthermore, we show that our extension for categorical features can lead to higher quality solutions.

We believe our work can be extended in a number of ways. Extending our formulations with additional constraints, such as minimum-support constraints or pruning constraints, or alternative optimization criteria is an interesting direction for future work. Investigating the impact of different tree structures and the use of our approach as part of an ensemble method are also interesting research directions. Finally, our approach for dealing with non-binary features can be extended to other classification models such as decision sets [25].

References

1 Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1418–1426, 2019.

2 Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3146–3153, 2020.

3 Florent Avellaneda. Efficient inference of optimal decision trees. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 3195–3202, 2020.

4 Kristin P Bennett. Global tree optimization: A non-greedy decision tree algorithm. *Journal of Computing Science and Statistics*, 26:156–160, 1994.

5 Jeremias Berg, Emir Demirović, and Peter J Stuckey. Core-boosted linear search for incomplete MaxSAT. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, pages 39–56. Springer, 2019.

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

7 Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 173–187. Springer, 2009.

8 Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

9 Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees.* Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

10 Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

11 Niklas Eén and Niklas Sörensson. Minisat SAT solver, 2003. URL: `http://minisat.se/Main.html`.

12 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 252–265. Springer, 2006.

13 Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, pages 1–28, 2021.

14 Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, 1996.

15 Hao Hu, Mohamed Siala, Emmanuel Hébrard, and Marie-José Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In *International Joint Conference on Artificial Intelligence and Pacific Rim International Conference on Artificial Intelligence (IJCAI-PRICAI)*, 2020.

16 Alexey Ignatiev, Joao Marques-Silva, Nina Narodytska, and Peter J Stuckey. Reasoning-based learning of interpretable ML models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, page in press, 2021.

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

18 Nina Narodytska, Alexey Ignatiev, Filipe Pereira, Joao Marques-Silva, and IS RAS. Learning optimal decision trees with SAT. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1362–1368, 2018.

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

20 OscaR Team. OscaR: Scala in OR, 2012. URL: `https://bitbucket.org/oscarlib/oscar`.

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

22 J Ross Quinlan. *C4. 5: programs for machine learning.* Elsevier, 2014.

23 Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25(3):226–250, 2020.

24 Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1625–1632, 2019.

25 Jinqiang Yu, Alexey Ignatiev, Peter J Stuckey, and Pierre Le Bodic. Computing optimal decision sets with sat. In *International Conference on Principles and Practice of Constraint Programming (CP)*, pages 952–970. Springer, 2020.