Learning Small Decision Trees with Large Domain

Eduard Eiben[1] , **Sebastian Ordyniak**[2] and **Giacomo Paesani**[2] and **Stefan Szeider**[3]
[1]Royal Holloway University of London, UK
[2]University of Leeds, UK
[3]Algorithms and Complexity Group, TU Wien, Vienna, Austria
eduard.eiben@rhul.ac.uk, {s.ordyniak, g.paesani}@leeds.ac.uk, sz@ac.tuwien.ac.at

Abstract

One favors decision trees (DTs) of the smallest size or depth to facilitate explainability and interpretability. However, learning such an optimal DT from data is well-known to be NP-hard. To overcome this complexity barrier, Ordyniak and Szeider (AAAI 21) initiated the study of optimal DT learning under the parameterized complexity perspective. They showed that solution size (i.e., number of nodes or depth of the DT) is insufficient to obtain fixed-parameter tractability (FPT). Therefore, they proposed an FPT algorithm that utilizes two auxiliary parameters: the maximum difference (as a structural property of the data set) and maximum domain size. They left it as an open question of whether bounding the maximum domain size is necessary.

The main result of this paper answers this question. We present FPT algorithms for learning a smallest or lowest-depth DT from data, with the only parameters solution size and maximum difference. Thus, our algorithm is significantly more potent than the one by Szeider and Ordyniak as it can handle problem inputs with features that range over unbounded domains. We also close several gaps concerning the quality of approximation one obtains by only considering DTs based on minimum support sets.

1 Introduction

Decision Trees (DTs) have proved to be extremely useful tools for describing, classifying, and generalizing data [Larose and Larose, 2014; Murthy, 1998; Quinlan, 1986]. Because of their simplicity, DTs are particularly attractive for providing interpretable models of the underlying data, an aspect whose importance has been strongly emphasized over recent years [Darwiche and Hirth, 2023; Doshi-Velez and Kim, 2017; Goodman and Flaxman, 2017; Lipton, 2018; Monroe, 2018]. In this context, one prefers *small trees* (trees of small size or small depth), as they are easier to interpret and require fewer tests to make a classification. Small trees are also preferred in view of the parsimony principle (Occam's Razor) since small trees are expected to generalize better to new data [Bessiere *et al.*, 2009]. However, learning small trees is computationally costly: it is NP-hard to decide whether a given data set can be represented by a DT of a certain size or depth [Hyafil and Rivest, 1976]. In view of this complexity barrier, several methods based on branch & bound algorithms, constraint programming, mixed-inter programming, or satisfiability solving have been proposed for learning small DTs [Avellaneda, 2020; Bessiere *et al.*, 2009; Aglin *et al.*, 2020a; Aglin *et al.*, 2020b; Bertsimas and Dunn, 2017; Demirovic *et al.*, 2022; Hu *et al.*, 2020; Janota and Morgado, 2020; Narodytska *et al.*, 2018; Shati *et al.*, 2021; Schidler and Szeider, 2021; Verhaeghe *et al.*, 2020; Verwer and Zhang, 2017; Verwer and Zhang, 2019; Zhu *et al.*, 2020]. This bulk of recent empirical work underlines the importance of computing optimal decision trees.

In this paper, we investigate the problem of finding small decision trees (w.r.t. size or depth) under the framework of *Parameterized Complexity* [Downey and Fellows, 2013; Gottlob *et al.*, 2002; Niedermeier, 2006]. This framework allows us to achieve a more fine-grained and qualitative analysis, revealing properties of the input data in terms of *problem parameters* that provide run-time guarantees for decision tree learning algorithms. The key notion of Parameterized Complexity is *fixed-parameter tractability (*FPT*)* which generalizes the classical polynomial time tractability by allowing the running time to be exponential in a function of the problem parameters while remaining polynomial in the input size (we provide more detailed definitions in Section 2). Fixed-parameter tractability captures the scalability of algorithms to large inputs as long as the problem parameters remain small. Several fundamental problems that arise in AI have been studied in terms of their fixed-parameter tractability, including Planning [Bäckström *et al.*, 2012], SAT and CSP [Bessière *et al.*, 2008; Gaspers *et al.*, 2017], Computational Social Choice [Bredereck *et al.*, 2017], Machine Learning [Ganian *et al.*, 2018], and Argumentation [Dvorák *et al.*, 2012].

For DT learning, we consider parameterizations of the following two fundamental NP-hard problems:

MINIMUM DECISION TREE SIZE (DTS): we are given a set of examples, labelled positive or negative, each over a set of features; each feature f ranges over a linearly ordered range of possible values (by choosing an arbitary ordering this also captures categorical data), and an integer s (for *size*). The task is to find a DT of minimum size or report correctly that no decision tree with at most s nodes exists. Here we

parameters		complexity
solution size	maximum difference	FPT †
solution size	-	W[2]-hard‡, in XP‡
-	maximum difference	para-NP-hard‡
-	-	para-NP-hard‡

Table 1: † this paper, Theorem 4; ‡ are results by Ordyniak and Szeider [2021].

consider DTs where each node tests whether a certain feature is below a certain threshold or not.

MINIMUM DECISION TREE DEPTH (DTD) is defined similarly, where instead of the bound s on the total number of nodes, a bound d (for *depth*) on the number of nodes on any root-to-leaf path is provided.

For both problems, it is natural to include the *solution size* (i.e., s for DTS and d for DTD, respectively) as a parameter since our objective is to learn DTs where these values are small. However, Ordyniak and Szeider's [2021] complexity analysis revealed that solution size is not sufficient to obtain fixed-parameter tractability. They, therefore, proposed two additional problem parameters: (i) the *maximum domain size*, i.e., the largest number of different values a feature ranges over, and (ii) the *maximum difference*, i.e., the largest number of features two examples with a different classification can disagree in. With these two additional parameters at hand, Ordyniak and Szeider could show that DTS and DTD are fixed-parameter tractable. They showed that without including the maximum difference in the parameterization, one loses fixed-parameter tractability. However, they left it open whether the maximum domain size is indeed needed as a parameter.

In this paper, we answer this open problem, obtaining fixed-parameter tractability of DTS and DTD just with the two parameters solution size and maximum difference. Our main result can be stated as follows.

- DTS *and* DTD *are fixed-parameter tractable parameterized by solution size and maximum difference (Theorem 4).*

This result completes Ordyniak and Szeider's parameterized complexity classification, as shown in Table 1.

Our result is surprising, as for similar problems, the domain size must be included in the parameterization. For instance, the Constraint Satisfaction Problem (CSP) is fixed-parameter tractable by the combined parameter primal treewidth and domain size [Gottlob *et al.*, 2002; Samer and Szeider, 2010], and by the combined parameter strong backdoor size and domain size [Gaspers *et al.*, 2017]; in both cases the problem becomes W[1]-hard (and hence fixed-parameter *in*tractable) when domain size is dropped from the parameterization.

Our result has a beneficial algorithmic impact. As we do not need to parameterize by maximum domain size, we have a significantly more powerful algorithm that allows us to compute optimal DTs (in terms of smallest depth/size) even for instances where features range over a large set of possible values. What makes our result further appealing is that the maximum difference, the only additional parameter we need in addition to solution size, is quite small in real-world data sets. Ordyniak and Szeider [2021] list values for various standard benchmark sets from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). In some cases, the maximum difference is two orders of magnitude smaller than the number of examples or features.

A subset of the features that suffices to correctly classify a classification instance is called a *support set*. Ordyniak and Szeider [2021] observed that, in general, a small or low-depth DT would not necessarily use a smallest support set. Indeed, this property of small or low-depth DTs provides a challenge to algorithmically finding such DTs, as we cannot first minimize the feature set in a preprocessing phase if we want to find DTs of the smallest size or lowest depth. In the second part of this paper, we quantify the impact on the size and depth of DTs when minimizing first the feature set. It turns out that regarding this question, it is significant whether the considered data is over features with unbounded domain size or if the domain size is bounded. For the unbounded domain case we obtain the following result.

- *The smallest size (depth) of a DT for a classification instance (with unbounded domain) using only features of a smallest support set can be arbitrarily larger than the size (depth) of an optimal DT for that classification instance (Theorem 17).*

For the bounded domain case (all features are binary), we obtain the following results.

- *The smallest size (depth) of a DT for a binary classification instance using only features of a smallest support set is at most by an exponential factor larger than the size (depth) of an optimal DT for that classification instance (Theorem 15).*
- *There exist binary classification instances where this exponential factor is unavoidable (Theorem 16).*

These separation results are relevant to practitioners who develop algorithms for DT minimization. It is tempting to first minimize the set of features to achieve a smaller instance size, so that the input to a SAT or CP encoding is easier to handle. However, our separation results establish that one has to consider that the result will be significantly worse than the optimum.

2 Preliminaries

We refer for a more in-depth treatment of Parameterized complexity (PC) to other sources [Cygan *et al.*, 2015]. PC considers problems in a two-dimensional setting, where a problem instance is a pair (I, k), where I is the main part and k is the parameter. A parameterized problem is *fixed-parameter tractable* if there exists a computable function f such that instances (I, k) can be solved in time $f(k)\|I\|^{O(1)}$.

2.1 Classification Problems

An *example* e is a function $e : \mathit{feat}(e) \to \mathbb{Z}$ defined on a finite set $\mathit{feat}(e)$ of *features*, where each feature f comes with a possibly infinite linearly ordered domain $\mathit{dom}(f) \subseteq \mathbb{Z}$, which we assume to be, w.l.o.g., a subset of the integers. For a set

E of examples, we put $\mathit{feat}(E) = \bigcup_{e \in E} \mathit{feat}(e)$. We say that two examples e_1, e_2 *agree* on a feature f if $f \in \mathit{feat}(e_1)$, $f \in \mathit{feat}(e_2)$ and $e_1(f) = e_2(f)$. If $f \in \mathit{feat}(e_1)$, $f \in \mathit{feat}(e_2)$ but $e_1(f) \neq e_2(f)$, we say that the examples *disagree on* f.

A *classification instance (CI)* (also called a *partially defined Boolean function* [Ibaraki *et al.*, 2011]) $E = E^+ \uplus E^-$ is the disjoint union of two sets of examples, where for all $e_1, e_2 \in E$ we have $\mathit{feat}(e_1) = \mathit{feat}(e_2)$. The examples in E^+ are said to be *positive*; the examples in E^- are said to be *negative*. A set X of examples is *uniform* if $X \subseteq E^+$ or $X \subseteq E^-$; otherwise X is *non-uniform*. We say that a CI E is binary if all features in $\mathit{feat}(E)$ are *binary*, i.e., $e(f) \in \{0, 1\}$ for every $f \in \mathit{feat}(e)$.

Given a CI E, a subset $F \subseteq \mathit{feat}(E)$ is a *support set* of E if any two examples $e_1 \in E^+$ and $e_2 \in E^-$ disagree in at least one feature of F. Finding a smallest support set, denoted by $\mathrm{MSS}(E)$, for a classification instance E is an NP-hard task [Ibaraki *et al.*, 2011, Theorem 12.2].

2.2 Decision Trees

A *decision tree* (DT) is a rooted binary tree T with vertex set $V(T)$ and edge set $A(T)$ such that each leaf node is either a *positive* or a *negative* leaf and the following holds for each non-leaf t of T:

- t is labelled with a feature denoted by $\mathit{feat}_T(t)$ or simply $\mathit{feat}(t)$ if T is clear from the context,
- t is labelled with an integer *threshold* denoted by $\lambda_T(t)$ or simply $\lambda(t)$ if T is clear from the context,
- t has 2 children, i.e., a left child and a right child.

We write $\mathit{feat}(T) = \{\, \mathit{feat}(t) \mid t \in V(T) \,\}$ for the set of all features used by T. The size of T is its number of nodes, i.e. $|V(T)|$.

Let E be a CI and let T be a DT with $\mathit{feat}(T) \subseteq \mathit{feat}(E)$. We say that a node t_A is a *left (right) ancestor* of t if t is contained in the subtree of T rooted at the left (right) child of t_A. For each node t of T, we define $E_T(t)$ as the set of all examples $e \in E$ such that for every left (right) ancestor t_A of t in T, it holds that $e(\mathit{feat}(t_A)) \leq \lambda(t_A)$ ($e(\mathit{feat}(t_A)) > \lambda(t_A)$). T *classifies* an example $e \in E$ if e is a positive (negative) example and $e \in E_T(l)$ for a positive (negative) leaf l of T. We say that T *classifies* E (or T *is a DT for* E) if T classifies all examples in E. See Figure 1 for an illustration of a CI and a DT that classifies E.

We will consider the following optimization problems.

E	f_1	f_2	f_3	f_4
$e_1 \in E^-$	0	5	1	-2
$e_2 \in E^-$	1	-1	3	0
$e_3 \in E^-$	1	0	-1	1
$e_4 \in E^-$	3	1	0	-1
$e_5 \in E^+$	4	-2	2	0
$e_6 \in E^+$	2	1	1	1

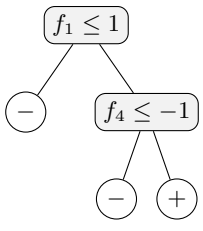

Figure 1: A CI $E = E^+ \uplus E^-$ with six examples and four features (left), a decision tree with 5 nodes that classifies E (right).

MINIMUM DECISION TREE SIZE (DTS)

Input: A CI E and an integer s.
Question: Find a DT of size at most s for E or report correctly that there is no DT for E of size at most s.

The problem MINIMUM DECISION TREE DEPTH (DTD) is defined very similarly only that there one is given an integer d (instead of s) and asks for a DT of depth at most d for E.

For two examples e and e' in E, we denote by $\delta(e, e')$ the set of features where e and e' disagree and we denote by $\delta_{\max}(E) = \max_{e^+ \in E^+ \wedge e^- \in E^-} |\delta(e^+, e^-)|$ the *maximum difference* between any non-uniform pair of examples.

Let T be a DT for E and $t \in V(T)$ be an inner node of T. We denote by T_t the (sub-)DT of T rooted at t. We say that t is *redundant* if either: (1) t is the root of T and either T_{c_ℓ} or T_{c_r} is a DT for E, where c_ℓ and c_r are the left and right children of t in T, or (2) t is the left (right) child of its parent p and t has a child c such that the tree obtained from T after removing T_c and t and making the other child of t the left (right) child of p is a DT for E. Intuitively, t is redundant if it is not required to distinguish any examples and can therefore be removed from T. We say that T is *non-redundant* if it does not contain any redundant node.

For the complexity analysis we set the *input size* $\|E\|$ of a CI E to $|E| \cdot (|\mathit{feat}(E)| + 1) \cdot \log D_{\max}$, where $D_{\max}$ is the maximum size of $\mathit{dom}(f)$ over all features f of E. We now give some simple auxiliary lemmas that are required by our algorithms.

Observation 1 ([Ordyniak and Szeider, 2021, Obs. 1])**.** *Let T be a DT for a CI E, then $\mathit{feat}(T)$ is a support set of E.*

Lemma 2 ([Ordyniak and Szeider, 2021, Cor. 9])**.** *Let E be a CI and let k be an integer. Then there is an algorithm that in time $\mathcal{O}(\delta_{\max}(E)^k |E|)$ enumerates all (of the at most $\delta_{\max}(E)^k$) minimal support sets of size at most k for E.*

Lemma 3 ([Ordyniak and Szeider, 2021, Lem. 5])**.** *Let A be a set of features of size a. Then the number of DTs without thresholds of size at most s that use only features in A is at most a^{2s+1} and those can be enumerated in $\mathcal{O}(a^{2s+1})$ time.*

3 FPT-algorithm

This section is devoted to a proof of our main result provided in the following theorem.

Theorem 4. DTS *and* DTD *are fixed-parameter tractable parameterized by the solution size and* $\delta_{\max}$.

To simplify the presentation and taking into account that the proof for DTD is almost identically to the proof for DTS, we will start by showing the result for DTS.

The overall structure of our algorithm is very similar to Algorithms 3 and 4 given in [Ordyniak and Szeider, 2021] and is illustrated in Algorithms 1 and 2. Namely, Algorithm 1 contains the main routine **minDT**, which given a CI E and an integer s outputs a minimum DT, i.e., a DT of minimum size, for E among all DTs of size at most s. To achieve this, the routine **minDT** first iterates over all minimal support sets of size at most s using Lemma 2. It then calls the routine

Algorithm 1 Main method for finding a DT of minimum size.

Input: CI E and integer s
Output: DT for E of minimum size (among all DTs of size at most s) if such a DT exists, otherwise `nil`

function MINDT(E, s)
  $\mathcal{S} \leftarrow$ "set of all minimal support sets for E of size at most s using Lemma 2"
  $B \leftarrow$ `nil`
  for $S \in \mathcal{S}$ **do**
    $T \leftarrow$ MINDTS(E, s, S)
    if ($T \neq$ `nil`) and ($B =$ `nil` or $|B| > |T|$) **then**
      $B \leftarrow T$
  if $B \neq$ `nil` and $|B| \leq s$ **then**
    return B
  return `nil`

Algorithm 2 Method for finding a DT of minimum size using at least the features in a given support set S.

Input: CI E, integer s, support set S for E with $|S| \leq s$
Output: DT of minimum size among all DTs T for E of size at most s such that $S \subseteq \mathit{feat}(T)$; if no such DT exists, `nil`

function MINDTS(E, s, S)
  $B \leftarrow$ "a minimum size DT for E of size at most s that uses exactly the features in S using Lemma 5"
  $H \leftarrow$ "a (S, s)-branching set $B(S, s)$ using Theorem 6"
  for $f \in H$ **do**
    $T \leftarrow$ MINDTS(E, s, $S \cup \{f\}$)
    if $T \neq$ `nil` and $|T| < |B|$ **then**
      $B \leftarrow T$
  if $|B| \leq s$ **then**
    return B
  return `nil`

minDTS, given in Algorithm 2, for every such minimal support set S to find a minimum DT T for E of size at most s such that $S \subseteq \mathit{feat}(T)$. Note that provided the correctness of **minDTS**, the correctness of **minDT** follows from Observation 1, because every DT for E must contain some minimal support set. Given E, s and a minimal support S, the routine **minDTS** computes a minimum DT T for E of size at most s such that $S \subseteq \mathit{feat}(T)$. The starting point (recursion start) of **minDTS** is the following lemma that allows to compute a minimum DT T for E of size at most s such that $S = \mathit{feat}(T)$.

Lemma 5 ([Ordyniak and Szeider, 2021, Theorem 4])**.** *Let E be a CI, $S \subseteq \mathit{feat}(E)$ be a support set for E, and let s be an integer. Then, there is an algorithm that runs in time $2^{\mathcal{O}(s^2)}\|E\|^{1+o(1)} \log \|E\|$ and computes a minimum DT among all DTs T with $\mathit{feat}(T) = S$ and $|T| \leq s$ if such a DT exists;otherwise* `nil` *is returned.*

After applying the above lemma to find a minimum DT T for E of size at most s such that $S = \mathit{feat}(T)$, the routine **minDTS** tries to find a minimum DT for E of size at most s that uses at least one feature outside of S. To achieve this the algorithm first computes a so-called (S, s)-branching set H, which informally is a "small" set of features such that every DT T for E of size at most s with $S \subsetneq \mathit{feat}(T)$ has to use at least one feature in H (see Subsection 3.1 for a formal definition of (S, s)-branching set). It then branches on every feature h in H and calls itself recursively for E, s, and $S \cup \{h\}$. The main ingredient of our algorithm compared to the algorithm given in [Ordyniak and Szeider, 2021], i.e., the FPT-algorithm for DTS if one additionally parameterizes by the maximum domain size of any feature, is the computation of the (S, s)-branching set, which we describe next.

3.1 Computing Branching Sets

Here, we will show that we can compute a small branching set, which is the main novel and crucial ingredient for our FPT-algorithm. Before we formally define branching sets, we need the following notions.

Let E be a CI. We denote by $\blacksquare$ a new feature, which we call the *unknown feature*, i.e., $\blacksquare \notin \mathit{feat}(E)$. A *DT pattern* is a DT T without thresholds that is allowed to use the unknown feature, i.e., $\mathit{feat}(T) \subseteq \mathit{feat}(E) \cup \{\blacksquare\}$. We say that an inner node t of T is *known* if $\mathit{feat}(t) \in \mathit{feat}(E)$ and *unknown* otherwise.

A DT pattern T' is an *extension* of a DT pattern T if $T = T'$ and $\mathit{feat}_{T'}(t) = \mathit{feat}_T(t)$ for every known node t of T. We say that T' is *complete* if $\mathit{feat}(T') \subseteq \mathit{feat}(E)$. A *threshold assignment* for a DT pattern T is a function $\lambda : \mathsf{KN}(T) \to \mathbb{Z}$ that provides a threshold assignment for every node of T in the set $\mathsf{KN}(T)$ of all known nodes of T.

In the following, let T be a DT pattern for a CI E. Note that we assume that if t is a node of T with $\mathit{feat}(t) = \blacksquare$, then any example that ends up in t is sent to both its left and its right child in T. In particular, we generalize $E_T(t)$ to DT patterns T with a threshold assignment λ by setting $E_T(t)$ to be the set of all examples $e \in E$ such that for every left (right) ancestor t_A of t in T, it holds that either $\mathit{feat}(t_A) = \blacksquare$ or $e(\mathit{feat}(t_A)) \leq \lambda(t_A)$ ($e(\mathit{feat}(t_A)) > \lambda(t_A)$).

We say that a node t of T is *valid* for a set $E' \subseteq E$ of examples if there is threshold assignment $\lambda : \mathsf{KN}(T) \to \mathbb{Z}$ such that either:

- t is a negative (positive) leaf of T and $E' \subseteq E^-$ ($E' \subseteq E^+$), or
- t is an unknown node of T and t has a child t' that is valid for E', or
- t is a known node of T with feature $f = \mathit{feat}(t)$ and the two children c_l and c_r of t in T are valid for $E'[f \leq \lambda(t)]$ and $E'[f > \lambda(t)]$, respectively.

We also say that T is *valid* for E' if the root r of T is valid for E'. Intuitively, T is valid for E' if it can be completed to a DT for E' that does not use of any of the unknown nodes.

Let E be a CI and let T be an invalid DT pattern for E. We say that a set $B \subseteq \mathit{feat}(E) \setminus \mathit{feat}(T)$ is a *branching set* for T if $B \cap (\mathit{feat}(T') \setminus \mathit{feat}(T)) \neq \emptyset$ for every proper extension T' of T that is valid for E. Let s be an integer and let S be a support set for E with $|S| \leq s$. We say that a set $B \subseteq \mathit{feat}(E) \setminus S$ is an *(S, s)-branching set* if $B \cap (\mathit{feat}(T) \setminus S) \neq \emptyset$ for every non-redundant DT T for E of size at most s with $S \subsetneq \mathit{feat}(T)$.

The remainder of this subsection is devoted to a proof of the following theorem, which constitutes the main novel technical contribution of this paper and we believe is interesting in its own right.

Theorem 6. *Let s be an integer, E be a CI and S be a support set for E with $|S| \leq s$. Then, an (S, s)-branching set of size*

at most $(s+3)^{2s+1}\delta_{\max}(E)$ *and can be computed in time* $\mathcal{O}((s+1)^{2s+1}2^{s^2/2}\|E\|^{1+o(1)}\log\|E\|)$.

The main ideas behind the proof of Theorem 6 are as follows. Given s, E, and S as defined in Theorem 6 our aim is to find a small set B of features, i.e., an (S,s)-branching set, such that $B \cap (\mathit{feat}(T) \setminus S) \neq \emptyset$ for every non-redundant DT T for E of size at most s such that $S \subsetneq \mathit{feat}(T)$. Let T be any such non-redundant DT for E, then replacing every feature in $\mathit{feat}(T) \setminus S$ with the new feature ■ and ignoring the threshold function gives rise to an invalid DT pattern T' for E; T' is invalid because T is non-redundant. The main ingredient behind our algorithm is now a routine that given any invalid DT pattern T' computes a small branching set for T'. Because an (S,s)-branching set can be obtained from the union of all branching sets for every possible invalid DT patterns for E of size at most s that uses only features in $S \cup \{■\}$, this now allows us to compute an (S,s)-branching set as follows. First we use the following corollary of Lemma 3 to enumerate all possible DT patterns T' for E of size at most s using only features in $S \cup \{■\}$.

Corollary 7. *Let A be a set of features of size a with ■ $\in A$. The number of DTs patterns of size at most s that use only features in A is at most a^{2s+1} and those can be enumerated in $\mathcal{O}(a^{2s+1})$ time.*

We then use Lemma 9 to decide whether T' is valid for E. Finally, if this not the case we use our routine to compute a branching set for T'. The (S,s)-branching set is then obtained as the union of all branching sets computed in this manner. Therefore, our main task now is to compute a branching set for a given invalid DT pattern for E.

Let E be a CI and let T be an invalid DT pattern for E. Our algorithm to compute a branching set for T proceeds in two main steps. First we compute a set EXP_t of expected examples for every node t of T, which intuitively contains all examples that: (1) will end up at t if no unknown node is replaced with a real feature and (2) is the smallest set of examples showing that T is invalid. Second, given EXP_t we compute an even smaller subset of examples, i.e., a so called pool set $P(r)$ for the root r of T, satisfying (1) and (2). We then show that any valid extension of T has to replace at least one unknown feature with a feature that distinguishes between two examples in the pool set. This then allows us to show that the set of all features $\bigcup_{e,e' \in P(r)} \delta(e,e')$ is a branching set for T. We start by showing how we compute the set of expected examples.

Computing the Set of Expected Examples

Let E be a CI and T be an invalid DT pattern for E. For every $t \in V(T)$, we define the set of *expected examples* EXP_t together with the *left and right thresholds*, denoted by $\lambda^L(t)$ and $\lambda^R(t)$, respectively, recursively as follows:

- if t is the root of T, then $\mathsf{EXP}_t = E$;
- if t is the left child of a known node p, then $\mathsf{EXP}_t = \mathsf{EXP}_p[f \leq \lambda^L(p)+1]$, where $f = \mathit{feat}(p)$ and $\lambda^L(p)$ is the maximum value in $\mathit{dom}(f)$ such that T_t is valid for $\mathsf{EXP}_t[f \leq \lambda^L(p)]$;
- if t is the right child of a known node p, then $\mathsf{EXP}_t = \mathsf{EXP}_p[f > \lambda^R(p)-1]$ where $f = \mathit{feat}(p)$ and $\lambda^R(p)$ is the minimum value in $\mathit{dom}(f)$ such that T_t is valid for $\mathsf{EXP}_t[f > \lambda^R(p)]$;
- if t is a child of an unknown node p, then $\mathsf{EXP}_t = \mathsf{EXP}_p$.

Before proving in Theorem 10 that we can efficiently compute EXP_t, $\lambda^L(t)$, and $\lambda^R(t)$ for every (fixed) node t of T, we need to show a simple but crucial property.

Lemma 8. *Let T be an invalid DT pattern for E. For every node t of T it holds that T_t is not valid for EXP_t.*

Algorithm 3

Input: CI E, DT pattern T for E
Output: `TRUE` if T is valid for E, `FALSE` otherwise

function ISVALID(E, T)
  $r \leftarrow$ "root of T"
  if r is a leaf **then**
    if r is negative (positive) and $E \subseteq E^-$ ($E \subseteq E^+$) **then**
      return `TRUE`
    return `FALSE`
  $c_\ell, c_r \leftarrow$ "left child and right child of r"
  if r is unknown **then**
    if ISVALID(E, T_{c_ℓ}) or ISVALID(E, T_{c_r}) **then**
      return `TRUE`
    return `FALSE`
  $f \leftarrow \mathit{feat}(r)$
  $(\lambda^L, \lambda^R) \leftarrow$ BINARYSEARCH(E, T, f, c_ℓ, c_r)
  if $\lambda^L \geq \lambda^R$ **then**
    return `TRUE`
  return `FALSE`

Algorithm 4 Algorithm to compute the pair $(\lambda^L(r), \lambda^R(r))$ for the root r of T

Input: CI E, DT pattern T, feature f of the root of T, left child c_ℓ of the root of T, right child c_r of the root of T
Output: the pair $(\lambda^L(r), \lambda^R(r))$

function BINARYSEARCH(E, T, f, c_ℓ, c_r)
  $D \leftarrow$ "array containing all elements in $\mathit{dom}_E(f)$ in ascending order"
  $L \leftarrow 0$; $R \leftarrow |D|-1$;
  while $L \leq R$ **do**
    $m \leftarrow \lfloor (L+R)/2 \rfloor$
    if ISVALID($E[f \leq D[m]], T_{c_\ell}$) **then**
      $L \leftarrow m+1$;
    else
      $R \leftarrow m-1$;
  $\lambda^L \leftarrow D[m-1]$ ▷ where $D[-1] = D[0]-1$
  $L \leftarrow 0$; $R \leftarrow |D|-1$;
  while $L \leq R$ **do**
    $m \leftarrow \lfloor (L+R)/2 \rfloor$
    if ISVALID($E[f > D[m]], T_{c_r}$) **then**
      $R \leftarrow m-1$;
    else
      $L \leftarrow m+1$;
  $\lambda^R \leftarrow D[m+1]$ ▷ where $D[|D|] = D[|D|-1]+1$
  return (λ^L, λ^R)

Algorithm 5 Algorithm to compute the triple $(\mathsf{EXP}_t, \lambda^L(t), \lambda^R(t))$ for every node $t \in V(T)$.

Input: CI E, DT pattern T
Output: the triple $(\mathsf{EXP}_t, \lambda^L(t), \lambda^R(t))$ for every node $t \in V(T)$.

function FINDLR(E, T)
  $r \leftarrow$ "root of T"
  if r is a leaf **then**
    return $(E, \texttt{nil}, \texttt{nil})$
  $c_\ell, c_r \leftarrow$ "left child and right child of r"
  if r is an unknown node **then**
    $O_\ell \leftarrow$ FINDLR(E, T_{c_ℓ})
    $O_r \leftarrow$ FINDLR(E, T_{c_r})
    return $(E, \texttt{nil}, \texttt{nil}) \cup O_\ell \cup O_r$
  $f \leftarrow \mathit{feat}(r)$
  $(\lambda^L, \lambda^R) \leftarrow$ BINARYSEARCH(E, T, f, c_ℓ, c_r)
  $O_\ell \leftarrow$ FINDLR($E[f \leq \lambda^L + 1], T_{c_\ell}$)
  $O_r \leftarrow$ FINDLR($E[f > \lambda^R - 1], T_{c_r}$)
  return $(E, \lambda^L, \lambda^R) \cup O_\ell \cup O_r$

The following lemma, which is a precursor for the computation of the expected examples in Theorem 10, is a relatively straightforward extension of [Ordyniak and Szeider, 2021, Lemma 6]; the algorithm behind the lemma is also illustrated in Algorithms 3 and 4.

Lemma 9. *Let E be a CI and T be a DT pattern of depth at most d. There is an algorithm with run-time $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$ deciding whether T is valid for E.*

Now we are finally ready to prove that we can efficiently compute EXP_t, $\lambda^L(t)$ and $\lambda^R(t)$ for every node $t \in V(T)$.

Theorem 10. *Let E be a CI, let T be a DT pattern of depth at most d. Then there is an algorithm that runs in time $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$ and computes the set EXP_t and thresholds $\lambda^L(t)$ and $\lambda^R(t)$ for every node $t \in V(T)$.*

Computing the Pool and Branching Set

Let E be a CI and T a DT pattern for E that is invalid for E and suppose that we have already computed the triple $(\mathsf{EXP}_t, \lambda^L(t), \lambda^R(t))$ for every node $t \in V(T)$. We say that T' is a *proper* extension of T if T' is an extension of T and $\mathit{feat}_{T'}(t) \notin \mathit{feat}(T)$ for every unknown node $t \in V(T)$ with $\mathit{feat}_{T'}(t) \neq \blacksquare$, i.e., unknown nodes of T that are known in T' are assigned to features not in $\mathit{feat}(T)$.

A *pool set* for T is a set P of examples such that for every proper extension of T that is valid for E there is a feature $f \in \mathit{feat}(T') \setminus \mathit{feat}(T)$ such that f distinguishes two examples in P. Let $P(t)$ be the set of examples defined recursively for every node t of T as follows: If t is a negative (positive) leaf node of T, then $P(t)$ contains any example in $E^+ \cap \mathsf{EXP}_t$ ($E^- \cap \mathsf{EXP}_t$). Note that such an example does always exists because of Lemma 8 and our assumption that T is invalid for E. Otherwise, t has a left child c_ℓ and a right child c_r and we set $P(t) = P(c_\ell) \cup P(c_r)$. Note that $P(t) \subseteq \mathsf{EXP}_t$ for every $t \in V(T)$. We show next that $P(T) = P(r)$ for the root r of T is a pool set for T.

Lemma 11. *Let E be a CI and T be an invalid DT pattern for E. Then, $P(T)$ is a pool set for T.*

Proof. Assume for a contradiction that this is not the case. Then, there is a proper extension T' of T that is valid for E such that no feature in $\mathit{feat}(T') \setminus \mathit{feat}(T)$ distinguishes between any two examples in $P(T)$. Let $\lambda : \mathsf{KN}(T') \to \mathbb{Z}$ be a threshold assignment for T' showing the validity of T'. We start by showing the following claim:

Claim 12. *Let t be an inner node of T such that $P(t) \subseteq E_{T'}(t)$, then t has a child c in T such that $P(c) \subseteq E_{T'}(c)$.*

Because the conditions of Claim 12 apply to the root r of T, it follows that T must have a leaf l with $P(l) \subseteq E_{T'}(l)$. But this implies that T'_l is not valid for $E_{T'}(l)$ a contradiction to our assumption that T' is valid for E. □

The next lemma shows that $P(T)$ is indeed small and can be computed efficiently.

Lemma 13. *Let E be a CI and T be an invalid DT pattern for E of height at most d. Then, $P(T) \leq 2^d$ and $P(T)$ can be computed in time $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$.*

Proof. $P(T) \leq 2^d$ follows because $|P(l)| = 1$ for every leaf of T and $|P(t)| = |P(c_\ell)| + |P(c_r)| = 2|P(c_\ell)|$ for every inner node t with children c_ℓ and c_r. To compute $P(T)$, we first use Theorem 10 to compute the triple $(\mathsf{EXP}_t, \lambda^L(t), \lambda^R(t))$ for every node $t \in V(T)$ in time $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$. We then compute $P(T)$ in a leaf-to-root manner in time $\big(|V(T)|\big)$. □

The next lemma now show that the set $B(T) = \bigcup_{e,e' \in P(T)} \delta(e, e')$ is a branching set for T, i.e., we can easily compute a branching set from a pool set.

Lemma 14. *Let E be a CI and T be an invalid DT pattern for E of height at most d. Then, $B(T)$ is a branching set for T of size at most $2^{2d}\delta_{\max}(E)$ and can be computed in time $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$.*

Proof. $B(T)$ is a branching set because $P(T)$ is a pool set for T due to Lemma 11. Moreover, because of Lemma 13, we have that $|B| \leq |P(T)|^2\delta_{\max}(E) \leq 2^{2d}\delta_{\max}(E)$ and the time required to compute $P(T)$ is $\mathcal{O}(2^{d^2/2}\|E\|^{1+o(1)} \log \|E\|)$, which dominates the time to compute $B(T)$. □

We are now ready to show Theorem 6.

Proof Sketch for Theorem 6. Here we only show that $B(S, s)$ is an (S, s)-branching set; the remainder of the proof can be found in the supplementary material. Let T be any non-redundant DT for E of size at most s such that $S \subsetneq \mathit{feat}(T)$ and let T' be the DT pattern for E obtained from T after setting $\mathit{feat}_{T'}(t) = \blacksquare$ for every $t \in V(T)$ with $\mathit{feat}_T(t) \notin S$ and ignoring all thresholds. Because T' has at least one unknown node and T is non-redundant, it follows that T' is invalid for E. Therefore, $T' \in \mathcal{T}$, which shows that $B(T') \subseteq B(S, s)$. Because $B(T')$ is a branching set for T' and T is a proper extension of T' that is valid for E, we obtain that $B(T') \cap (\mathit{feat}(T) \setminus S) \neq \emptyset$ and therefore $B(S, s)$ is an (S, s)-branching set, as required. □

We are now ready to show Theorem 4, i.e., that DTS is fixed-parameter tractable parameterized by size and $\delta_{\max}$.

Proof Sketch of Theorem 4. Our algorithm for DTS is illustrated in Algorithm 1 and Algorithm 2. In this proof sketch, we only show the correctness of the algorithm. So suppose that there is a DT for E of size at most s that uses all features in S and let T be any such DT of minimum size. Because the algorithm returns a DT of minimum size among all the DTs that it considers, it suffices to show that the algorithm considers T. Even stronger we will show that the algorithm considers all DTs T' for E of size at most s such that $\mathit{feat}(T') = \mathit{feat}(T)$.

Towards showing the correctness of Algorithm 1, consider the case that E has a DT of size at most s and let T be such a DT of minimum size. Because of Observation 1, $\mathit{feat}(T)$ is a support set for E and therefore $\mathit{feat}(T)$ contains a minimal support set S of size at most s. Because the algorithm (Line 4 of Algorithm 1) iterates over all minimal support sets of size at most s for E, it follows that Algorithm 2 is called with parameters E, s, and S.

If $\mathit{feat}(T) = S$, then the algorithm finds a DT for E of size at most $|T|$ in Line 2 of Algorithm 2 because of Lemma 5. If, on the other hand, $\mathit{feat}(T) \setminus S \neq \emptyset$, then $H \cap \mathit{feat}(T) \neq \emptyset$, where H is the (S, s)-branching set computed in Line 3 of Algorithm 2; this is because H is an (S, s)-branching set and T is a non-redundant (since minimal) DT for E of size at most s such that $S \subsetneq \mathit{feat}(T)$. Therefore, the function **minDTS** is called recursively for parameters E, s, and $S \cup \{f\}$, where f is an arbitrary feature in $\mathit{feat}(T) \cap H$. From now onward the argument repeats and eventually the function **minDTS** is called with parameters E, s, and $\mathit{feat}(T)$ at which point the algorithm finds a DT for E of size at most $|T|$ in Line 2 of Algorithm 2. Finally, it is easy to see that if Algorithm 1 outputs a DT T, then it is a valid solution. This is because T must have been computed in Line 2 of Algorithm 2, which implies that T is a DT for E. Moreover, T has size at most s, because of Line 8 in Algorithm 1. □

4 Approximation Using Support Sets

Given Observation 1 it is tempting to think that it sufficies to consider only DTs that use the features from some minimal support set. Indeed, if this were the case, then our FPT-algorithm from the previous section could be significantly simplified, i.e., it would no longer be necessary to find branching sets as it would suffice to enumerate all minimal support sets with the help of Lemma 2. Unfortunately, Ordyniak and Szeider [2021] showed that this is not the case and the difference between an optimal DT and an optimal DT that is only allowed to employ features from some minimal support set can be arbitrarily high at least in absolute terms even for binary CIs. Nevertheless, it was left open whether and how well the simple approach using only minimal support sets can be exploited to obtain good approximate solutions for DTS and DTD and this is what we will explore in this section. In particular, let $\mathsf{opt}^s(E)$ and $\mathsf{opt}^d(E)$ be the minimum size respectively depth of a DT for a CI E and let $\mathsf{opt}^s_{\mathrm{SS}}(E)$ and $\mathsf{opt}^d_{\mathrm{SS}}(E)$ the minimum size respectively depth of a DT for E that is only allowed to use the features from some minimal support set. Because $\mathsf{opt}^s_{\mathrm{SS}}(E)$ and $\mathsf{opt}^d_{\mathrm{SS}}(E)$ can be computed using a much simpler algorithm that requires only Lemma 2 and Lemma 5, we want to explore whether they can be used to approximate $\mathsf{opt}^s(E)$ and $\mathsf{opt}^d(E)$.

As a starting point consider the case of binary CIs. In particular, let E be a binary CI and let S be a minimum support set for E. Then, because of Observation 1 any DT for E has size at least $|S|$ and depth at least $\log |S|$. Moreover, E has a DT of size at most $2^{|S|+1}$ and depth at most $|S| + 1$, i.e., the complete DT using only the features in S. Therefore, we obtain the following theorem showing that $\mathsf{opt}^s_{\mathrm{SS}}$ and $\mathsf{opt}^d_{\mathrm{SS}}$ approximate opt^s and opt^d, respectively.

Theorem 15. *Let E be a binary CI. Then, $\mathsf{opt}^s_{\mathrm{SS}}(E) \leq 2^{\mathsf{opt}^s(E)}$ and $\mathsf{opt}^d_{\mathrm{SS}}(E) \leq 2^{\mathsf{opt}^d(E)}$.*

As our main novel result in this section (for binary CIs), we show next that the ratios obtained in Theorem 15 are indeed best possible and therefore no better approximation for DTS and DTD can be obtained by considering only DTs that merely use the features of some minimal support set.

Theorem 16. *For every integer $k \geq 1$, there is a binary CI L_k such that $\mathsf{opt}^s(L_k) \leq 2k + 5$ and $\mathsf{opt}^s_{\mathrm{SS}}(L_k) \geq 2^{k+1} - 1$. Similarly, there is a binary CI L^d_k such that $\mathsf{opt}^d(L^d_k) \leq \log(k) + 2$ and $\mathsf{opt}^d_{\mathrm{SS}}(L^d_k) \geq k + 1$.*

Finally, we consider the case of non-binary CIs and show the following theorem, which essentially rules out any approximation algorithm based solely on minimal support sets.

Theorem 17. *For every integer $n \geq 1$, there are a CIs L_n and L^d_n such that $\mathsf{opt}^s(L_n), \mathsf{opt}^d(L^d_n) \leq 5$ and $\mathsf{opt}^s_{\mathrm{SS}}(L_n), \mathsf{opt}^d_{\mathrm{SS}}(L^d_n) \geq n$.*

5 Conclusion

We have established novel results that contribute to the foundations of learning interpretable machine learning models. Our main result is algorithmic. We have devised a parameterized algorithm that allows us to efficiently learn an optimal DT (with the smallest number of nodes or lowest depth). This answers an open question by Ordyniak and Szeider [2021], who had to include the maximum domain size for their FPT result and completes their complexity classification for DT learning. As pointed out in the introduction, our result stands out because for similar problems (like the CSP), the inclusion of domain size is inevitable.

Our second result deals with the question of what one loses when working with a smallest set of features (a minimum support set) when learning a DT of a small size or depth. It turns out that this question strongly depends on whether the domain size is bounded or not. We show that the gap between the optimal solution and one that depends on the smallest set of features can be arbitrarily large for the unbounded domain case. For the bounded domain case, the gap can be bounded by an exponential function, and that this bound is tight. This result is of interest to practitioners as it is a natural approach for heuristics to perform feature reduction before learning the DT.

Acknowledgements

Stefan Szeider acknowledges support by the Austrian Science Fund (FWF, project P32441) and the Vienna Science and Technology Fund (WWTF, project ICT19-065). Giacomo Paesani and Sebastian Ordyniak acknowledge support from the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).

References

[Aglin *et al.*, 2020a] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. *Proc. AAAI 2020*, pages 3146–3153, 2020.

[Aglin *et al.*, 2020b] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. PyDL8.5: a library for learning optimal decision trees. *Proc. IJCAI 2020*, pages 5222–5224, 2020.

[Avellaneda, 2020] Florent Avellaneda. Efficient inference of optimal decision trees. *Proc. AAAI 2020*, pages 3195–3202, 2020.

[Bäckström *et al.*, 2012] Christer Bäckström, Yue Chen, Peter Jonsson, Sebastian Ordyniak, and Stefan Szeider. The complexity of planning revisited - a parameterized analysis. *Proc. AAAI 2012*, pages 1735–1741, 2012.

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

[Bessière *et al.*, 2008] Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. The parameterized complexity of global constraints. *Proc. AAAI 2008*, pages 235–240, 2008.

[Bessiere *et al.*, 2009] Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. *Proc. CP 2009*, 5732:173–187, 2009.

[Bredereck *et al.*, 2017] Robert Bredereck, Jiehua Chen, Rolf Niedermeier, and Toby Walsh. Parliamentary voting procedures: Agenda control, manipulation, and uncertainty. *Journal of Artificial Intelligence Research*, 59:133–173, 2017.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Darwiche and Hirth, 2023] Adnan Darwiche and Auguste Hirth. On the (complete) reasons behind decisions. *Journal of Logic, Language and Information*, 32(1):63–88, 2023.

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

[Doshi-Velez and Kim, 2017] Finale Doshi-Velez and Been Kim. A roadmap for a rigorous science of interpretability. *CoRR*, abs/1702.08608, 2017.

[Downey and Fellows, 2013] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer Verlag, 2013.

[Dvorák *et al.*, 2012] Wolfgang Dvorák, Sebastian Ordyniak, and Stefan Szeider. Augmenting tractable fragments of abstract argumentation. *Artificial Intelligence*, 186:157–173, 2012.

[Ganian *et al.*, 2018] Robert Ganian, Iyad Kanj, Sebastian Ordyniak, and Stefan Szeider. Parameterized algorithms for the matrix completion problem. *Proc. ICML 2018*, pages 1642–1651, 2018.

[Gaspers *et al.*, 2017] Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivný. Backdoors into heterogeneous classes of SAT and CSP. *Journal of Computer and System Sciences*, 85:38–56, 2017.

[Goodman and Flaxman, 2017] Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, 2017.

[Gottlob *et al.*, 2002] Georg Gottlob, Francesco Scarcello, and Martha Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002.

[Hu *et al.*, 2020] Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. *Proc. IJCAI 2020*, pages 1170–1176, 2020.

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

[Ibaraki *et al.*, 2011] Toshihide Ibaraki, Yves Crama, and Peter L. Hammer. *Partially defined Boolean functions (in Boolean Functions: Theory, Algorithms, and Applications)*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.

[Janota and Morgado, 2020] Mikolás Janota and António Morgado. Sat-based encodings for optimal decision trees with explicit paths. *Proc. SAT 2020*, 12178:501–518, 2020.

[Larose and Larose, 2014] Daniel T. Larose and Chantal D. Larose. *Discovering Knowledge in Data: An Introduction to Data Mining*. John Wiley & Sons, 2nd edition, 2014.

[Lipton, 2018] Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*, 61(10):36–43, 2018.

[Monroe, 2018] Don Monroe. AI, explain yourself. *AI Communications*, 61(11):11–13, 2018.

[Murthy, 1998] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[Narodytska *et al.*, 2018] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva. Learning optimal decision trees with SAT. *Proc. IJCAI 2018*, pages 1362–1368, 2018.

[Niedermeier, 2006] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2006.

[Ordyniak and Szeider, 2021] Sebastian Ordyniak and Stefan Szeider. Parameterized complexity of small decision tree learning. *Proc. AAAI 2021*, pages 6454–6462, 2021.

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

[Samer and Szeider, 2010] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *Journal of Computer and System Sciences*, 76(2):103–114, 2010.

[Schidler and Szeider, 2021] André Schidler and Stefan Szeider. SAT-based decision tree learning for large data sets. *Proc. AAAI 2021*, pages 3904–3912, 2021.

[Shati *et al.*, 2021] Pouya Shati, Eldan Cohen, and Sheila A. McIlraith. SAT-based approach for learning optimal decision trees with non-binary features. *Proc. CP 2021*, 210(50):1–16, 2021.

[Verhaeghe *et al.*, 2020] Hélène Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. *Constraints*, 25(3-4):226–250, 2020.

[Verwer and Zhang, 2017] Sicco Verwer and Yingqian Zhang. Learning decision trees with flexible constraints and objectives using integer optimization. *Proc. CPAIOR 2017*, 10335:94–103, 2017.

[Verwer and Zhang, 2019] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. *Proc. AAAI 2019*, pages 1625–1632, 2019.

[Zhu *et al.*, 2020] Haoran Zhu, Pavankumar Murali, Dzung T. Phan, Lam M. Nguyen, and Jayant Kalagnanam. A scalable MIP-based method for learning optimal multivariate decision trees. *Proc. NeurIPS 2020*, 2020.