On Computing Optimal Tree Ensembles

Christian Komusiewicz [*1] **Pascal Kunz** [*2 3] **Frank Sommer** [*1] **Manuel Sorge** [*4]

Abstract

Random forests and, more generally, (decision-)tree ensembles are widely used methods for classification and regression. Recent algorithmic advances allow to compute decision trees that are optimal for various measures such as their size or depth. We are not aware of such research for tree ensembles and aim to contribute to this area. Mainly, we provide two novel algorithms and corresponding lower bounds. First, we are able to carry over and substantially improve on tractability results for decision trees, obtaining a $(6\delta DS)^S \cdot \mathrm{poly}$-time algorithm, where S is the number of cuts in the tree ensemble, D the largest domain size, and δ is the largest number of features in which two examples differ. To achieve this, we introduce the witness-tree technique which also seems promising for practice. Second, we show that dynamic programming, which has been successful for decision trees, may also be viable for tree ensembles, providing an $\ell^n \cdot \mathrm{poly}$-time algorithm, where ℓ is the number of trees and n the number of examples. Finally, we compare the number of cuts necessary to classify training data sets for decision trees and tree ensembles, showing that ensembles may need exponentially fewer cuts for increasing number of trees.

1. Introduction

Random forests is a method for classification or regression in which we construct an ensemble of decision trees for (random subsets of) the training data and, in the classification phase, aggregate their outcomes by majority voting. The random-forests method has received a tremendous amount of attention for its simplicity and improved accuracy over plain decision trees (Breiman, 2001; Verikas et al., 2011; Kulkarni & Sinha, 2013; Rokach, 2016). Commonly, fast heuristics without performance guarantees are used for computing random forests (Kulkarni & Sinha, 2013; Rokach, 2016), in particular for computing the individual decision trees in the forest. For plain decision trees, researchers lately made several advances in computing optimal decision trees, that is, decision trees that provably optimize criteria such as minimizing the tree size (Bessiere et al., 2009; Narodytska et al., 2018; Carrizosa et al., 2021; Demirović et al., 2022). With that increased amount of attention also came theoretical advances, showing the limits and opportunities for developing efficient exact algorithms for computing decision trees (Ordyniak & Szeider, 2021; Kobourov et al., 2022; Eiben et al., 2023). One impetus to computing optimal decision trees is that minimizing the size reduces tendencies to overfitting (Bessiere et al., 2009; Demirović et al., 2022). It is conceivable that such benefits transfer to globally optimizing the tree ensembles computed by random forests. However, apart from sporadic hardness results (Tamon & Xiang, 2000), we are not aware of exact algorithmic research for tree ensembles. In this work, we aim to initiate this direction; that is, we begin to build the theoretical footing for exact algorithmics of computing optimal tree ensembles and provide potential avenues for exact algorithms that are guaranteed to provide optimal results with acceptable worst-case running times.

We study the algorithmic properties of two canonical formulations of the training problem for tree ensembles: We are given a set of training examples labeled with two classes and a number ℓ of trees and we want to compute a tree ensemble containing ℓ trees that classifies the examples consistently with the given class labels.[1] We want to minimize either the sum of the tree sizes, resulting in the problem MINIMUM TREE ENSEMBLE SIZE (MTES), or the largest size of a tree in the ensemble, resulting in the

[*]Equal contribution [1]Fakultät für Mathematik und Informatik, Friedrich-Schiller-Universität Jena, Germany [2]Algorithmics and Computational Complexity, TU Berlin, Germany [3]Algorithm Engineering, HU Berlin, Germany [4]Institute of Logic and Computation, TU Wien, Austria. Correspondence to: Christian Komusiewicz <c.komusiewicz@uni-jena.de>, Pascal Kunz <p.kunz.1@tu-berlin.de>, Frank Sommer <frank.sommer@uni-jena.de>, Manuel Sorge <manuel.sorge@ac.tuwien.ac.at>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]To keep the presentation focused, we consider mainly the case without training error. See the conclusion for extensions to minimizing training error.

problem MINIMAX TREE ENSEMBLE SIZE (MMAXTES).[2] Both contain as a special case the problem of computing a minimum-size decision tree, which is known to be NP-hard (Hyafil & Rivest, 1976; Ordyniak & Szeider, 2021). However, the hardness constructions do not necessarily reflect practical data. Thus, we are interested in precisely which properties make the problems hard or tractable.

Mainly, we provide two novel algorithms for MTES and MMAXTES[3] and matching lower-bound results for their running times. We call the first one *witness-tree algorithm*. This algorithm demonstrates that prospects for tractable algorithms for optimizing decision trees can be non-trivially generalized to optimizing tree ensembles. Namely, it was known that for small tree size s, moderate maximum domain size D of any feature, and moderate number δ of features in which two examples differ[4], a minimum decision tree can be computed efficiently, that is, in $f(s, D, \delta) \cdot \mathrm{poly}$ time, where poly is a polynomial in the input size (Ordyniak & Szeider, 2021). However, the function f is at least $\delta^s \cdot (D^s 2\delta)^s \cdot 2^{s^2}$ and the algorithm involves enumerative steps in which the worst-case running time equals the average case. We show that, even for the more general MTES, we can improve the running time to $\mathcal{O}((6\delta DS)^S \cdot S\ell n)$, where S denotes the sum of the tree sizes, ℓ the number of trees in the ensemble, and n the number of training examples (Theorem 4.1). Moreover, we can avoid the enumerative approach, obtaining a search-tree algorithm that is both conceptually simpler and more easily amenable to heuristic improvements such as early search-termination rules and data reduction.[5] We achieve this by growing the trees iteratively and labeling their leaves with witness examples that need to be classified in these leaves. This allows us to localize misclassifications and their rectification, shrinking the search space. We believe that this technique may have practical applications beyond improving the worst-case running times as we do here. The running time that we achieve is tight in the sense that we cannot decrease the exponent to $o(S)$ without violating reasonable complexity-theoretic assumptions (Theorem 4.4).

Recently, exponential-time dynamic programming has been applied to compute optimal decision trees and the resulting trees have shown comparable performance to (heuristic) random forests on some datasets (Demirović et al., 2022). With the second algorithm that we provide, we investigate the potential of dynamic programming for computing optimal tree ensembles. We first show that minimizing decision trees can be done in $\mathcal{O}(3^n)$ time, where n is the number of input examples, by a dynamic-programming approach that works on all possible splits of the examples (Corollary 5.2). (Indeed, the algorithm employed by Demirović et al. (2022) similarly computes a table over all possible splits in the worst case.) We then extend this algorithm to tree ensembles with ℓ trees, achieving $(\ell + 1)^n \cdot \mathrm{poly}$ running time (Theorem 5.3). Unfortunately, we also show that the running time cannot be substantially improved: A running time of $f(\ell) \cdot 2^{o(\log \ell) \cdot n}$ would violate reasonable complexity-theoretic assumptions (Theorem 5.4).

Finally, we compare the power of decision trees and tree ensembles in terms of their sizes. Here, we show that a training data set $\mathcal{D}$ that can be classified by a tree ensemble with ℓ trees of size at most s, can also be classified by a decision tree of size $(s + 1)^\ell$ (Theorem 3.1). However, such an exponential increase is necessary in the worst case: We show that there are such training data sets $\mathcal{D}$ that cannot be classified by any decision tree of size roughly $(s/2)^{\ell/2}$ (Theorem 3.2).

In summary, as the number of trees in a tree ensemble grow, the classification power increases exponentially over decision trees. Nevertheless, we are able to carry over and substantially improve on tractability results for decision trees if in particular the number of cuts in the optimal ensemble is relatively small. The underlying witness-tree technique seems promising to try in practice. Furthermore, we show that dynamic programming, which has been successful for decision trees, may also be viable for tree ensembles. We also provide matching lower bounds for the running times. Apart from tuning our algorithms, in the future, deconstructing these lower bounds may provide further guidelines towards which properties of the input data we may exploit for efficient algorithms and which we likely may not.

Proofs of statements marked with ★ are deferred to a full version of the paper.

2. Preliminaries

For $n \in \mathbb{N}$ we use $[n] \coloneqq \{1, 2, \ldots, n\}$. For a vector $x \in \mathbb{R}^d$ we denote by $x[i]$ the ith entry of x.

Let Σ be a set of class symbols; unless stated otherwise, we use $\Sigma = \{\mathsf{blue}, \mathsf{red}\}$. A decision tree in $\mathbb{R}^d$ with classes Σ is the following. Let T be an ordered binary tree, that is, each inner node has a well-defined left and right child. Let $\mathsf{dim}\colon V(T) \to [d]$ and $\mathsf{thr}\colon V(T) \to \mathbb{R}$ be labelings of each internal node $t \in V(T)$ by a *dimension* $\mathsf{dim}(t) \in [d]$ and a *threshold* $\mathsf{thr}(t) \in \mathbb{R}$. Furthermore, let $\mathsf{cla}(\ell)\colon V(T) \to \Sigma$

[2] It is also natural to consider the depths of the trees instead of their sizes, but results are usually transferable between these two optimization goals and the size makes the presentation more accessible.

[3] The algorithms work on the decision version of these problems, but they easily apply to the optimization versions as well.

[4] See Ordyniak & Szeider (2021) for measurements showing that this is a reasonably small parameter in several datasets.

[5] In the meantime since this paper has been accepted, Eiben et al. (2023) showed that for minimum-size decision trees the dependency on the domain size can be dropped, that is, there is an $f(s, \delta) \cdot \mathrm{poly}$-time algorithm. It still uses enumerative steps which would be infeasible in practice.

be a labeling of the leaves of T by class symbols. Then the tuple $(T, \mathsf{dim}, \mathsf{thr}, \mathsf{cla})$ is a *decision tree* in $\mathbb{R}^d$ with classes Σ. We often omit the labelings $\mathsf{dim}, \mathsf{thr}, \mathsf{cla}$ and just refer to the tree T. The *size* of T is the number of its internal nodes. We conveniently call the internal nodes of T and their associated labels *cuts*.

A *training data set* is a tuple (E, λ) of a set of *examples* $E \subseteq \mathbb{R}^d$ and their class labeling $\lambda\colon E \to \Sigma$. Given a training data set, we fix for each dimension i a minimum-size set $\mathsf{Thr}(i)$ of thresholds that distinguishes between all values of the examples in the ith dimension. In other words, for each pair of elements e and e' with $e[i] < e'[i]$, there is at least one value $t \in \mathsf{Thr}(i)$ such that $e[i] < t < e'[i]$. Let $t \in \mathbb{R}$ be some threshold. We use $E[f_i \leq t] = \{x \in E \mid x[i] \leq t\}$ and $E[f_i > t] = \{x \in E \mid x[i] > t\}$ to denote the set of examples of E whose ith dimension is less or equal and strictly greater than the threshold t, respectively. Now let T be a decision tree. Each node $t \in V(T)$, including the leaves, defines a subset $E[T, t] \subseteq E$ as follows. For the root t of T, we define $E[T, t] := E$. For each non-root node t, let p denote the parent of t. We then define $E[T, t] := E[T, p] \cap E[f_{\mathsf{dim}(p)} \leq \mathsf{thr}(p)]$ if t is the left child of p and $E[T, t] := E[T, p] \cap E[f_{\mathsf{dim}(p)} > \mathsf{thr}(p)]$ if t is the right child of p. If the tree T is clear from the context, we simplify $E[T, t]$ to $E[t]$. We say that T *classifies* (E, λ) if for each leaf $u \in V(T)$ and each example $e \in E[u]$ we have $\lambda(e) = \mathsf{cla}(u)$ (recall that cla is a labeling of the leaves of T by classes). Note that the set family that contains $E[u]$ for all leaves u of T forms a partition of E. Thus for each example $e \in E$ there is a unique leaf u such that $e \in E[u]$. We also say that u is *e's leaf*. We say that $\mathsf{cla}(u)$ is the class *assigned* to e by T and we write $T[e]$ for $\mathsf{cla}(u)$.

A *tree ensemble* is a set of decision trees. A tree ensemble $\mathcal{T}$ *classifies* (E, λ) if for each example $e \in E$ the majority vote of the trees in $\mathcal{T}$ agrees with the label $\lambda(e)$. That is, for each example $e \in E$ we have $\lambda(e) = \arg\max_{\sigma \in \Sigma} |\{T \in \mathcal{T} \mid T[e] = \sigma\}| =: \mathcal{T}[e]$. To avoid ambiguity in the maximum, we fix an ordering of Σ and break ties according to this ordering. If $\Sigma = \{\mathsf{blue}, \mathsf{red}\}$ we break ties in favor of blue. The *overall size* of a tree ensemble $\mathcal{T}$ is the sum of the sizes of the decision trees in $\mathcal{T}$.

The computational problems that we consider are as follows.

> MINIMUM TREE ENSEMBLE SIZE (MTES)
> *Instance:* A training data set (E, λ), a number ℓ of trees, and a size bound S.
> *Question:* Is there a tree ensemble of overall size at most S that classifies (E, λ) and contains exactly ℓ trees?

When restricting to $\ell = 1$, MTES is known as MINIMUM DECISION TREE SIZE (DTS) (Ordyniak & Szeider, 2021; Kobourov et al., 2022). In the variant MINIMAX TREE ENSEMBLE SIZE (MMAXTES), instead of S, we are given an integer s and we ask whether there is a tree ensemble that classifies (E, λ) and contains exactly ℓ trees, each of which has size at most s.

Our analysis is within the framework of parameterized complexity (Gottlob et al., 2002; Flum & Grohe, 2006; Niedermeier, 2006; Cygan et al., 2015; Downey & Fellows, 2013). Let $L \subseteq \Sigma^*$ be a computational problem specified over some alphabet Σ and let $p\colon \Sigma^* \to \mathbb{N}$ be a parameter, that is, p assigns to each instance of L an integer parameter value (which we simply denote by p if the instance is clear from the context). We say that L is *fixed-parameter tractable* (FPT) with respect to p if it can be decided in $f(p) \cdot \mathrm{poly}(n)$ time where n is the input encoding length. The corresponding hardness concept related to fixed-parameter tractability is W[t]-hardness, $t \geq 1$; if problem L is W[t]-hard with respect to p then L is thought to not be fixed-parameter tractable; see (Flum & Grohe, 2006; Niedermeier, 2006; Cygan et al., 2015; Downey & Fellows, 2013) for details. The Exponential Time Hypothesis (ETH) (Impagliazzo & Paturi, 2001; Impagliazzo et al., 2001) states that 3SAT on n-variable formulas cannot be solved in $2^{o(n)}$ time.

3. Decision Trees Versus Tree Ensembles

We will call a decision tree and a tree ensemble *equivalent* if any training data set is classified by the one if and only if it is classified by the other. We start by comparing the minimum size of a decision tree to that of a minimum-size decision tree ensemble, showing that there are examples where the latter is significantly smaller. Vidal & Schiffer (2020, Theorem 2) obtained similar results for the depth of a decision tree and decision tree ensemble, showing that any training data set that can be classified by a decision tree ensemble with ℓ trees of depth t can also be classified by a tree with depth $\ell \cdot t$ and that this bound is tight. Here, we analyze trees and tree ensembles in terms of their size, showing that a minimum ensemble can be exponentially smaller than any equivalent decision tree.

Theorem 3.1 (★). *Any training data set that can be classified by a decision tree ensemble consisting of ℓ trees, each of size at most s, can also be classified by a decision tree of size $(s+1)^\ell - 1$.*

Theorem 3.2 (★). *For any odd $\ell, s \in \mathbb{N}$, there is a training data set that can be classified by a decision tree ensemble containing ℓ trees of size s each, but cannot be classified by a single decision tree of size smaller than*

$$\frac{(s+1)^\ell}{\ell\left(\frac{s+1}{2}\right)^{\frac{\ell-1}{2}}} - 1.$$

Proof. For any $x \in \mathbb{N}^\ell$, let $\mathrm{even}(x) := \{i \in [\ell] \mid x[i] \text{ is even}\}$ and $\mathrm{odd}(x) := [\ell] \setminus \mathrm{even}(x)$. Furthermore, let $\mathrm{ev}(x)$ and $\mathrm{od}(x)$ denote the sizes of $\mathrm{even}(x)$ and $\mathrm{odd}(x)$, respectively. Let $E := \{x \in [s+1]^\ell \mid |\mathrm{ev} - \mathrm{od}| = 1\}$ and $\lambda\colon E \to \{\mathsf{blue}, \mathsf{red}\}$ with $\lambda(x) = \mathsf{blue}$ if and only if $\mathrm{ev}(x) > \mathrm{od}(x)$. We show that (E, λ) fulfills the claim.

We defer the proof that there is a decision tree ensemble $\mathcal{T}$ with ℓ trees of size at most s that classifies (E, λ) to the full version.

We will show that any decision tree T that classifies E has at least $\frac{(s+1)^\ell}{\ell(\frac{s+1}{2})^{\frac{\ell-1}{2}}}$ leaves. Observe that $|E| \geq \binom{\ell}{\frac{\ell+1}{2}}(\frac{s+1}{2})^\ell \geq \frac{2^\ell}{\ell}(\frac{s+1}{2})^\ell = \frac{(s+1)^\ell}{\ell}$, because even if we fix some subset I of $[\ell]$ such that $x[i]$ is to be even if and only if $i \in I$, then there are $\frac{s+1}{2}$ possible values for each component of x. Therefore, it is sufficient to prove that $|E[T,t]| \leq (\frac{s+1}{2})^{\frac{\ell-1}{2}}$ for every leaf t of T.

Let t be a leaf of T. Without loss of generality, assume that $\mathsf{cla}(t) = \mathsf{blue}$, that is $\lambda(x) = \mathsf{blue}$ for all $x \in E[T,t]$. We will show that $x[i] = y[i]$ for all $x, y \in E[T,t]$ and $i \in \mathrm{even}(x)$. Suppose that $x[i] \neq y[i]$, $i \in \mathrm{even}(x)$, and $x, y \in E[T,t]$. Without loss of generality, $x[i] < y[i]$. Define $z \in [s+1]^\ell$ by $z[i] := x[i]+1$ and $z_j := x_j$ for all $j \in [\ell] \setminus \{i\}$. Then, $\mathrm{even}(z) = \mathrm{even}(x) \setminus \{i\}$, implying that $\lambda(z) = \mathsf{red}$. Hence, $z \notin E[T,t]$. This implies that in T there must be a node v with $\dim(v) = i$ and $\mathsf{thr}(v) = x[i]$ on the path from the root to t. However, this means that $y \notin E[T,t]$. Hence, the examples in $E[T,t]$ can differ only in the components in $\mathrm{odd}(x)$. Moreover, $\mathrm{od}(x) = \frac{\ell-1}{2}$ and $[s+1]$ contains $\frac{s+1}{2}$ odd values.

Since the size of a binary tree is the number of leaves minus one, the claim follows. □

This result still leaves a considerable gap between the upper and lower bound. We conjecture that the lower bound can be improved: for example, by showing that in the example presented in the proof of Theorem 3.2 the number of examples in each leaf is, on average, smaller than we showed.

4. The Witness-Tree Algorithm

In this section, we prove the following theorem. Recall that S is the desired overall size of the tree ensemble, s is the maximum size of a tree in the ensemble, ℓ is the number of trees in the ensemble, D is the largest domain of a feature, δ is the largest number of features in which two examples of different classes differ, and n is the number of training examples.

Theorem 4.1. MINIMUM TREE ENSEMBLE SIZE *can be solved in* $\mathcal{O}((6\delta DS)^S \cdot S\ell n)$ *time and* MINIMAX TREE ENSEMBLE SIZE *in* $\mathcal{O}(2^\ell \cdot (\delta D\ell(2s+1))^{s\ell} \cdot s\ell^2 n)$ *time.*

The basic idea is to start with a tree ensemble that contains only trivial trees and to successively refine the trees in the ensemble until all input examples are classified correctly. To facilitate the refinement process, each leaf of each tree is assigned a distinct example, called witness. In a recursive process we then aim to find refinements of the trees such that each witness is classified in the assigned leaf. This will speed up the refinement process because it enables us to detect examples that need to be cut away from witnesses in some trees of the ensemble.

Formally, a *witness tree* is a tuple $(T, \mathsf{dim}, \mathsf{thr}, \mathsf{cla}, \mathsf{wit})$ wherein $(T, \mathsf{dim}, \mathsf{thr}, \mathsf{cla})$ is a decision tree and $\mathsf{wit}\colon V(T) \to E$ is a mapping from the leaves of T to the set of examples such that for each leaf t we have $\mathsf{wit}[t] \in E[t]$. The images of wit are called *witnesses*. Note that a witness is not necessarily classified correctly, that is, we permit $T[\mathsf{wit}(t)] \neq \lambda(\mathsf{wit}(t))$. A *witness ensemble* is a set of witness trees.

We aim to successively refine the trees in a witness ensemble until all examples are classified correctly. For this, an example $e \in E$ is *dirty* for some tree T (or tree ensemble $\mathcal{F}$) if the label $T(e)$ (or $\mathcal{F}(e)$) assigned to e by T (or by $\mathcal{F}$) is not equal to $\lambda(e)$.

Next, we define a refinement of a tree in an ensemble. All our refinements will take a dirty example and change the class label assigned to this example by one of the decision trees. Consider a witness tree T and a dirty example e for T. Intuitively, we take the leaf t of T in which e is classified and consider its witness $\mathsf{wit}(t)$. Then we pick a way of introducing into T a new cut on the path from the root to t that cuts apart $\mathsf{wit}(t)$ and e. This then results in a refinement of T.

Formally, let T be a witness tree. A *one-step refinement* R of T is a witness tree constructed in one of the following two ways (illustrated in Figure 1):

Possibility one: Add a new root r to T, labeled with a dimension $\mathsf{dim}(r)$ and threshold $\mathsf{thr}(r)$, put the old root of T to be the left or right child of r, and put the other child of r to be a new leaf v, labeled with an arbitrary class label and with a witness $x \in E$ such that $x \in E[R, v]$.

Possibility two: Pick any edge f in T. Subdivide f with a new node u, labeled with a dimension $\mathsf{dim}(u)$ and threshold $\mathsf{thr}(u)$, and add a new leaf v as a child to u, labeled with an arbitrary class label and with a witness $x \in E$. The order of the children of u is chosen such that $x \in E[R, v]$.

This finishes the definition of a one-step refinement. We also say that the one-step refinement *introduces* the new leaf v, the witness x, and the node r or u (thought of as the nodes including their associated labelings), respectively. Observe that the refinement is a witness tree and thus the previous witnesses need to be preserved. That is, the choices of

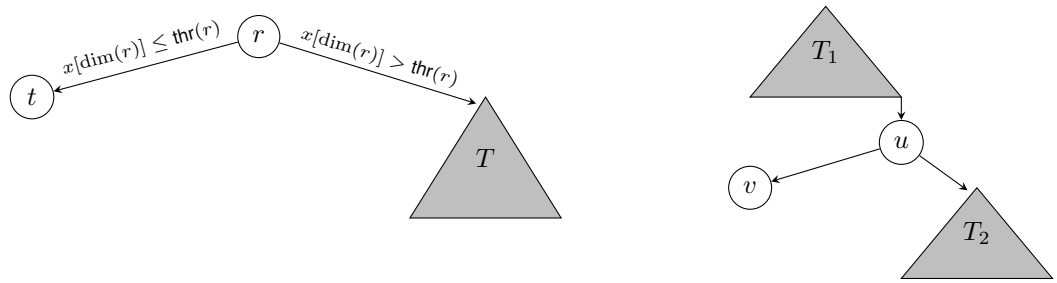

Figure 1. Two ways of refining a tree: On the left an new root r and a new leaf t are introduced. On the right, an existing edge between the subtrees T_1 and T_2 is subdivided with a vertex u and a new leaf v is introduced.

the dimension $\mathsf{dim}(r), \mathsf{dim}(u)$ and threshold $\mathsf{thr}(r), \mathsf{thr}(u)$ need to be such that each witness is still classified in its leaf. In formulas, for each leaf t of T it must hold that $\mathsf{wit}(t) \in E[R, t]$.

A *refinement* R of a witness tree T is obtained by a series $T = T_1, T_2, \ldots, T_k = R$ of witness trees such that T_i is a one-step refinement of T_{i-1}. A decision tree R (without witness labeling) is a *refinement* of a witness tree T if there is a labeling of the leaves of R by witnesses such that (a) the labeling results in a witness tree, that is, for each leaf t of R the witness $\mathsf{wit}(t)$ of t is in $E[R, t]$, and (b) after the labeling the witness tree R is a refinement of T. If a tree ensemble $\mathcal{C}'$ consists of the trees of a witness ensemble $\mathcal{C}$ or refinements thereof, then we say $\mathcal{C}'$ is a *refinement* of $\mathcal{C}$.

For the correctness proof for our algorithm we need a property of refinements that shows that the order in which we introduce nodes for certain refinements is immaterial.

Lemma 4.2. *Let T_1 be a witness tree, T_2 a one-step refinement of T_1, and T_3 a one-step refinement of T_2. Let T_3 introduce an inner node u. Assume that T_3 does not subdivide the edge incident with a new leaf introduced in T_2. Then there is a one-step refinement S_2 of T_1 and a one-step refinement S_3 of S_2 such that S_2 introduces u and $S_3 = T_3$.*

Proof. Refinement T_3 may introduce a new root or subdivide an edge. Suppose that T_3 introduces u as a new root above the root r of T_2. If r is also present in T_1, then T_2 subdivides an edge f of T_1. Thus, in S_2 we may instead introduce the root u, maintaining that edge f is present, and then in S_3 we may subdivide the edge f as in T_2. This results in the same witness tree S_3 as T_3 and thus maintains the sequence of refinements. If r is not present in T_1, then T_2 introduces the new root r above the old root r' of T_1. Instead, we may proceed as follows. In S_2 we first introduce the new root u above the old root r' of T_1. Then, in S_3 we subdivide the edge between r and u to introduce the node r' with the same labels as in T_2. Again, this results in the same witness tree S_3 as T_3.

Now suppose that T_3 subdivides the edge f to introduce node u. If f is present in T_1, then T_2 introduces a new root that is not incident with f or subdivides an edge different from f. In both cases, we may reverse the two operations, resulting in the same tree T_3. If f is not present in T_1, then f is a new edge introduced in T_2. By precondition, f is not incident with a new leaf of T_2. Hence, f is introduced by adding a new root above an old root or by subdividing an edge. If f was introduced by adding a new root r into T_1, we may instead proceed as follows. To obtain S_2, we take T_1 and introduce u as a new root. Afterwards, to obtain S_3 we take S_2 and introduce r as a new root above u. We again have that T_3 and S_3 are the same and the sequence of refinements is maintained. Suppose that f was introduced in T_2 by subdividing an edge f' of T_1 to introduce some node v. Then, in T_3 node v is the parent of u or vice-versa. Without loss of generality, assume that v is the parent of u. Instead, we may first subdivide f' in T_1 and introduce u to obtain S_2 and then subdivide the edge from u to its parent in S_2 to introduce v and obtain S_3. This again results in the same witness tree as T_3. Concluding, we may replace T_3 and T_2 with S_3 and S_2, obtaining the required properties. □

We can now describe the recursive algorithm for solving MTES. The pseudo-code is given in Algorithm 1. As mentioned, it checks whether the current witness ensemble is sufficiently small and classifies the input and, if so, reports it as a solution. Otherwise, it finds a dirty example e and tries all possibilities of reclassifying the example in a refinement of one of the trees in the current ensemble. Note that since e is dirty, a tree which can be refined always exists. Then it continues recursively. In Line 6, a one-step refinement of T is *important* if it is obtained by introducing a new node w that is labeled by a dimension $i \in [d]$ in which e and the witness x of e's leaf in T differ, i.e., $e[i] \neq x[i]$, and by a threshold $\delta \in \mathsf{Thr}(i)$ such that δ is between $e[i]$ and $x[i]$.

Note that, if $\ell > S$, then necessarily in the solution ensemble there are trivial trees without any cuts, that is, trees that classify all examples as blue or all examples as red. Now with a factor ℓ in the running time, we may determine how

Algorithm 1: Computing tree ensembles.

1 **Function** `RefineEnsemble` $(\mathcal{C}, (E, \lambda), S)$
  Input: A witness ensemble $\mathcal{C}$, a training data set (E, λ), and a size threshold $S \in \mathbb{N}$.
  Output: A tree ensemble of overall size at most S that is a refinement of $\mathcal{C}$ and classifies (E, λ) or $\perp$ if none exists.
2 **if** *overall size of $\mathcal{C}$ is larger than S* **then return** $\perp$
3 **if** *$\mathcal{C}$ classifies (E, λ)* **then return** $\mathcal{C}$
4 $e \leftarrow$ a dirty example for $\mathcal{C}$
5 **for** *each tree T in $\mathcal{C}$ in which e is not a witness* **do**
6 **for** *each important one-step refinement T' of T introducing e as witness and s.t. $T'[e] = \lambda(e)$* **do**
7 $\mathcal{C}' \leftarrow \mathcal{C}$ with T replaced by T'
8 $\mathcal{D} \leftarrow$ `RefineEnsemble` $(\mathcal{C}', (E, \lambda), S)$
9 **if** $\mathcal{D} \neq \perp$ **then return** $\mathcal{D}$
10 **return** $\perp$

many such trivial trees of either type there are (by trying all possibilities). Thus, in the following, we will assume that each tree has at least one inner node and thus that $\ell \leq S$.

The initial calls to `RefineEnsemble` are made with the following 2^ℓ witness ensembles $\mathcal{C}$: For each tree T in $\mathcal{C}$ we pick an arbitrary distinct example and try both possibilities for whether e is classified as $\lambda(e)$ or not (i.e. with the other class) and make T to be a tree consisting of a single leaf labeled by the corresponding class and with e as its witness. This concludes the description of the algorithm. For the algorithm for MMAXTES we replace S by s and we modify the check in Line 2 to check that the size of the largest tree is larger than s instead.

Proof of Theorem 4.1. We now show that the algorithm described above achieves the required running time and that it is correct.

For the running time, observe that, after one of the 2^ℓ initial calls, the algorithm describes a search tree in which each node corresponds to a call to `RefineEnsemble`. The depth of this tree is at most S for MTES and at most $s\ell$ for MMAXTES because in each call at least one refinement is made and thus the overall size increases by at least one. We claim that each search-tree node has at most $\delta D(2S + \ell)$ or $\delta D(2s+1)\ell$ children, respectively. To see this, we show that the total number of refinements of $\mathcal{C}$ considered in Line 6 is bounded by that number: Each such refinement is specified (1) by a new root or an edge on a root-leaf path of a tree in $\mathcal{C}$, (2) a dimension in the newly introduced node, and (3) a threshold in the newly introduced node.

For (3) there are at most D possibilities.

For (2) there are at most δ possibilities: Observe that e has a different class label than the witness w of its old leaf. Thus, there are at most δ dimensions in which e and w differ.

For (1) there are at most ℓ ways to choose to introduce a new root. Furthermore, since each edge in a tree is incident with an inner node and each inner node u is incident with at most two edges to children of u, there are at most $2S$ ways (resp. $2s\ell$ ways) to choose an edge for subdivision.

Thus, the overall search tree has size at most $(\delta D(2S + \ell))^S \leq (3\delta DS)^S$ (resp. $(\delta D(2s + 1)\ell)^{s\ell}$). Accounting for the $2^\ell \leq 2^S$ initial calls and noticing that the operations in one search-tree node take $\mathcal{O}(Sn)$ time yields the claimed running time.

It remains to prove the correctness. Clearly, if the algorithm returns something different from $\perp$, then it is a tree ensemble that classifies (E, λ) and is of the required size. Now assume that there is a tree ensemble that classifies (E, λ) and is of the required size. We show that the algorithm will not return $\perp$. We say that a witness-tree ensemble $\mathcal{C}$ is *good* if there is a tree ensemble $\mathcal{C}^\star$ that classifies (E, λ) of the required size, and such that $\mathcal{C}^\star$ refines $\mathcal{C}$. We claim that (1) one of the ensembles $\mathcal{C}$ of an initial call to `RefineEnsemble` is good and (2) that if $\mathcal{C}$ in a call to Algorithm 1 is good, then either it classifies (E, λ) or in at least one recursive call `RefineEnsemble`$(\mathcal{C}', \cdot, \cdot)$ it is the case that $\mathcal{C}'$ is good. Observe that it is enough to prove both claims.

As to claim (1): Consider a solution $\mathcal{C}^\star$ and the witnesses that were chosen arbitrarily for $\mathcal{C}$ before the initial calls to `RefineEnsemble`. Fix an arbitrary mapping of trees between $\mathcal{C}$ and $\mathcal{C}^\star$. Consider a tree T in $\mathcal{C}$ and its corresponding tree $T^\star$ in $\mathcal{C}^\star$. Observe that $T^\star$ has a leaf t such that $E[T^\star, t]$ contains the (single) witness of T. Label t with this witness. For each remaining leaf, pick an arbitrary example that is classified in this leaf and label it as its witness. (Note that, without loss of generality, there is at least one example in each leaf because if there is a leaf without an example then we can find a smaller solution ensemble.) Observe that the witness tree resulting from $T^\star$ by this labeling is a refinement of T if the leaves containing the single witness of T have the same class label. Since we try all possible class labels for the leaf in T before the initial calls to `RefineEnsemble`, eventually we obtain that $T^\star$ is a refinement of T and indeed this holds for all pairs of mapped trees of $\mathcal{C}$ and $\mathcal{C}^\star$. Thus, $\mathcal{C}$ is good.

As to claim (2): Assume that the witness-tree ensemble $\mathcal{C}$ is good and let $\mathcal{C}^\star$ be a corresponding witness-tree ensemble. If $\mathcal{C}$ classifies (E, λ) then there is nothing to prove. Otherwise, there is at least one dirty example for $\mathcal{C}$. Let e be the dirty example picked by the algorithm in Line 4. Consider the classes assigned to e by the trees in $\mathcal{C}$ and those assigned by the corresponding refinements in $\mathcal{C}^\star$. Observe that there

is at least one tree $T \in \mathcal{C}$ such that its refinement $R \in \mathcal{C}^\star$ assigns a different class to e. In a refinement, the class of a witness is never changed and thus e is not a witness in T. Hence, the for loop in Line 5 selects the tree T in one iteration.

We now claim that the loop in Line 6 will select a refinement T' such that R (thought of as a non-witness decision tree) is a refinement of T'. Consider a sequence of one-step refinements $T = T_1, T_2, \ldots, T_k = R$. We call a refinement T_j in this sequence *e-rerouting* if T_j introduces a new leaf t such that $e \in E[T_j, t]$. We claim that we may assume that T_2 is e-rerouting. For a contradiction, assume that in all sequences $(T_i)_{i\in[k]}$ the first e-rerouting one-step refinement T_j satisfies $j > 2$. (Note that such a refinement always exists because e is classified differently in R from T.) Pick the sequence $(T_i)_{i\in[k]}$ such that j is minimal. Let T_j introduce some node w. Observe that e's leaf t of T is present in T_j and that there is a path from w to t. Thus, w is not introduced by subdividing the edge incident with a leaf that has been introduced in T_{j-1}. Thus, by Lemma 4.2 we may replace T_j and T_{j-1} by one-step refinements that result in the same witness tree T_j and such that T_{j-1} introduces w instead. This is a contradiction to the minimality of j and thus we may assume that T_2 is e-rerouting.

Next, we claim that we may assume that the witness assigned to the new leaf introduced in T_2 is e. Before we can show this, we need to ensure that there are no further e-rerouting refinements in the sequence. Consider an e-rerouting refinement T_j after T_2, introducing some node u and a new leaf t such that $e \in E[t]$. Consider the other child v of u. By Lemma 4.2 we may assume that all nodes in the subtree of T_j rooted at v are introduced after u and thus that u has the two leaves t and v as children. Thus, instead of introducing t in T_j we may instead introduce v. That is, we may replace T_j by a different refinement that is not e-rerouting, maintaining the sequence of refinements. After doing this for all e-rerouting refinements it is the case that the leaf t containing e is introduced once in T_2 and then maintained in every refinement until we reach R. Thus, we may relabel the witness of t to be e in all the refinements. Finally, observe that we may assume that T_2 is important and thus T_2 equals T', the refinement selected by the algorithm in Line 6. Thus, the refined ensemble $\mathcal{C}'$ constructed from T' is good, as claimed. □

Recall that MINIMUM DECISION TREE SIZE (DTS) is the special case of MTES in which $\ell = 1$ (and thus without loss of generality $s = S$). Thus we have the following.

Corollary 4.3. MINIMUM DECISION TREE SIZE *can be solved in $\mathcal{O}((6\delta Ds)^s \cdot sn)$ time.*

This improves on the running time for DTS given by Ordyniak & Szeider (2021) (see their main theorem, Theorem 8).

The following theorem shows that the exponent in our running time cannot be improved.

Theorem 4.4 (★). *Solving* MTES *in $(\delta DS)^{o(S)} \cdot \mathrm{poly}$ time would contradict the Exponential Time Hypothesis, even if $D = 2$ and $\ell = 1$.*

5. Tight Exponential-Time Algorithm

We now give an algorithm that solves MTES in $(\ell + 1)^n \cdot \mathrm{poly}$ time, where $n := |E|$ is the number of examples. This running time is single-exponential in n for every fixed number of trees. More importantly, we show that this running time is essentially optimal. To obtain the algorithm, we first show how to compute in a suitable running time the sizes of smallest trees for essentially all possible classification outcomes of a decision tree.

Lemma 5.1. *Given a training data set (E, λ) one can compute in $3^n \cdot \mathrm{poly}$ time for all $E' \subseteq E$ the size of a smallest decision tree T such that $T[e] =$ blue if and only if $e \in E'$.*

Proof. We solve the problem using dynamic programming over subsets of E. The dynamic-programming table has entries of the type $Q[E^b, E^r]$ where E^b and E^r are example sets. Each table entry stores the size of a smallest decision tree on $E^b \cup E^r$ where exactly the examples of E^b receive the label blue. We fill the table entries for increasing values of $|E^b \cup E^r|$.

We initialize the table by setting

$$Q[E^b, \emptyset] := 0 \text{ and } Q[\emptyset, E^r] := 0$$

for all $E^b \subseteq E$ and all $E^r \subseteq E$. This is correct since in these cases, a decision tree without cuts and only one leaf with the appropriate class label suffices.

The recurrence to fill the table when E^b and E^r are nonempty is

$$Q[E^b, E^r] := \min_{i\in[d], t\in \mathsf{Thr}(i)} Q[E^b[f_i \le t], E^r[f_i \le t]] + Q[E^b[f_i > t], E^r[f_i > t]] + 1.$$

Recall that $\mathsf{Thr}(i)$ denotes some minimum-size set of thresholds that distinguishes between all values of the examples in the ith dimension. In other words, for each pair of elements e and e' with $e[i] < e'[i]$, there is at least one value $t \in \mathsf{Thr}(i)$ such that $e[i] < t < e'[i]$. Moreover, we only consider those cases where $E^b[f_i \le t] \cup E^r[f_i \le t] \neq \emptyset$ and $E^b[f_i > t] \cup E^r[f_i > t] \neq \emptyset$. That is, we consider only the case that the cut gives two nonempty subtrees. This ensures that the recurrence only considers table entries with smaller set sizes.

The idea behind the recurrence is that we consider all possibilities for the cut at the root and then use the smallest

decision trees to achieve the desired labeling for the two resulting subtrees. The size of the resulting tree is the size of the two subtrees plus one, for the additional root vertex. The formal correctness proof is standard and omitted.

The running time bound can be seen as follows. The number of table entries is $\mathcal{O}(3^n)$ since each entry corresponds to a 3-partition of E into E^b, E^r, and $E \setminus (E^b \cup E^r)$. Each entry can be evaluated in polynomial time since the number of dimensions and the size of $\mathsf{Thr}(i)$ is polynomial in the input size. □

The above lemma directly gives an algorithm for DTS with the same running time.

Corollary 5.2. MINIMUM DECISION TREE SIZE *can be solved in* $3^n \cdot \mathrm{poly}$ *time.*

Before proving the running time bound for constructing tree ensembles, let us remark that the above algorithm can be used not only to produce perfect decision trees but also to give a set of Pareto-optimal trees for the trade-off between size and essentially any type of efficiently computable classification error, for example precision, recall, or F_1-score.

We now use this algorithm as a subroutine in an algorithm for MINIMUM TREE ENSEMBLE SIZE.

Theorem 5.3. *For* $\ell > 1$*, one can solve* MINIMUM TREE ENSEMBLE SIZE *in* $(\ell+1)^n \cdot \mathrm{poly}$ *time.*

Proof. We use again dynamic programming. It is not sufficient to use subsets of elements that are classified correctly. Instead, we build the solution by iteratively adding trees and storing for each example e how often e is classified correctly.

To store subsolutions, we use a table R with entries of the type $R[c, j]$ where c is a length-n integer vector where each c_i is an integer in $[0, \lceil \ell/2 \rceil]$ for each $i \in [n]$ and $j \in [\ell]$. An entry $R[c, j]$ stores the smallest total size of any set of j decision trees such that each element e_i is classified correctly exactly c_i times if $c_i < \lceil \ell/2 \rceil$ and at least c_i times if $c_i = \lceil \ell/2 \rceil$. The distinction between $c_i < \lceil \ell/2 \rceil$ and $c_i = \lceil \ell/2 \rceil$ allows us to assign only one value of c_i to the situation that e_i is already correctly classified irrespective of the other trees of the ensemble.

The first step of the algorithm is to compute for all $E' \subseteq E$ the smallest size of any decision tree T assigning the blue label exactly to all $e \in E'$. From this information, we can directly compute for all $E' \subseteq E$, the size of a smallest decision tree that classifies all examples in E' correctly and all examples in $E \setminus E'$ incorrectly. We will store these sizes in table entries $Q[E']$.

Now, we initialize R for $j = 1$, by setting

$$R[c, 1] := \begin{cases} Q[E'] & \exists E' \subseteq E : c = \mathbb{1}_{E'}, \\ +\infty & \text{otherwise.} \end{cases}$$

Here, we let $\mathbb{1}_{E'}$ denote the indicator vector for E'. Now, for $j > 1$, we use the recurrence

$$R[c, j] := \min_{E' \subseteq E, c' : c' \oplus \mathbb{1}_{E'} = c} R[c', j-1] + Q[E']. \quad (1)$$

Here, $\oplus$ is a truncated addition, that is, for the ith component of c', we add 1 if it is strictly smaller than $\lceil \ell/2 \rceil$. If for some $R[c, j]$ the minimum ranges over an empty set, then we set $R[c, j] := +\infty$.

The idea of the recurrence is simply that the jth tree classifies some element set E' correctly and that this increases the number of correct classifications for all elements of E'. The smallest size of a tree ensemble with ℓ trees to correctly classify E can then be found in $R[c^*, \ell]$ where we let c^* denote the length-n vector where each component has value $\lceil \ell/2 \rceil$. The formal proof is again standard and omitted.

The table has size $(\lceil \ell/2 \rceil + 1)^n \cdot \ell$. The bottleneck in the running time to fill the table is the time needed for evaluating the $\min$ in Equation 1. A straightforward estimation gives a time of $2^n \cdot \lceil \ell/2 \rceil^n$ for each entry since we consider all possible subsets E' of E and possibly all vectors c'. Instead, we may fill the table entries also in a forward direction, that is, for each c' and each $E' \subseteq E$, we compute $c' \oplus \mathbb{1}_{E'}$ and update the table entry for $R[c, j]$ if $R[c', j-1] + Q[E']$ is smaller than the current entry of $R[c, j]$. This way, the total time for updating table entries is $2^n \cdot \lceil \ell/2 \rceil^n \leq (\ell+1)^n$ since we consider 2^n possible choices for E' at each vector c' and directly derive the corresponding c for each choice. The overall time bound follows from the observation that the $3^n \cdot \mathrm{poly}$ time needed for the preprocessing is upper-bounded by $(\ell+1)^n$ since $\ell > 1$. □

We now show that, under a standard conjecture in complexity theory, this running time cannot be improved substantially.

We show this by a reduction from MULTICOLORING. Here, one is given a graph G and two integers a and x, and wants to assign each vertex in $V(G)$ a set of b out of a colors such that each two adjacent vertices receive different color sets. MULTICOLORING cannot be solved in $f(x) \cdot x^{o(n)}$ time, where n is the number of vertices even if $a = \Theta(x^2 \log x)$ unless the ETH fails (Bonamy et al., 2019). Observe that in a solution for MULTICOLORING, the vertices having some color c form an independent set in G. In our reduction, we have a *choice dimension* for each maximal independent set of G. Furthermore, we have a *vertex example* for each vertex of G. Also, we set $\ell := 2a + 1$ and $S := 2a + 1$. The main idea of our reduction is that there are exactly a many trees cutting a choice dimension such that each vertex example is

correctly classified in at least x of these trees. We achieve this by adding some *dummy dimensions* and further vertex examples such that each correct tree ensemble consists of exactly ℓ trees having exactly one inner node, as we show.

Theorem 5.4 (★). *Solving* MINIMUM TREE ENSEMBLE SIZE *in* $f(\ell) \cdot 2^{o(\log \ell) \cdot n}$ *time would contradict the Exponential Time Hypothesis.*

Now, Theorem 5.4 implies that the running time $(\ell + 1)^n \cdot \mathrm{poly}$ of the algorithm in Theorem 5.3 cannot be significantly improved, unless the ETH is wrong.

Our proof of Theorem 5.4 also implies hardness for the larger parameter S and that this hardness holds even if $D = 2$, that is, each feature is binary.

Corollary 5.5. *Solving* MINIMUM TREE ENSEMBLE SIZE *on instances with binary features in* $f(S) \cdot 2^{o(\log S) \cdot n}$ *time would contradict the Exponential Time Hypothesis.*

Our proof of Theorem 5.4 also implies hardness for MMAXTES. For this result the proof is simpler, since no argument is needed that each tree in the ensemble has exactly one inner node.

Corollary 5.6. *Solving* MINIMAX TREE ENSEMBLE SIZE *on instances with binary features and* $s = 1$ *in* $f(\ell) \cdot 2^{o(\log \ell) \cdot n}$ *time would contradict the ETH.*

6. Outlook

We conclude by mentioning a few avenues for possible future research.

- In Theorem 4.1, we showed that MINIMUM TREE ENSEMBLE SIZE and MINIMAX TREE ENSEMBLE SIZE are both fixed-parameter tractable when parameterized by δ (the maximum number of dimensions in which a pair of examples differ), D (the domain size), S and s (the total tree ensemble size and the maximum size of a tree in the ensemble, respectively), and ℓ (the number of trees in the ensemble), respectively. It would be interesting to investigate the problem for strictly smaller parameterizations. Of course, lower bounds for MINIMUM DECISION TREE SIZE also apply to MTES and MMAXTES. Hence, these two problems are $W[2]$-hard with respect to (D, s, ℓ) and (D, S, ℓ), respectively, and NP-hard for constant values of (δ, D, ℓ). This leaves the parameterized complexity of MTES for (δ, S, ℓ) and (δ, D, S) and of MMAXTES for (δ, s, ℓ) for (δ, D, s) as open problems.

- Theorem 4.1 naturally raises the question of whether those problems admit polynomial-size kernels for that parameterization. Intuitively, a polynomial-size kernel is a polynomial-time algorithm that takes an instance for the problem at hand as input and shrinks this instance to a size that is bounded by a polynomial in the parameter. For details on kernelization, we refer to (Fomin et al., 2019; Cygan et al., 2015). Essentially, a kernelization is simply a polynomial-time preprocessing procedure which comes with a provable guarantee that the preprocessed instance is small relative to the parameter. Even preprocessing algorithms without a guarantee could be of practical interest.

- The decision tree and tree ensemble models could be extended in several ways: For instance, one could dispense with the restriction that cuts must be axis-aligned and allow cuts along any hyperplane. Lower bounds for constructing optimum trees in this model were proven by Goodrich et al. (1995) and Grigni et al. (2000), but the parameterized complexity of this problem has not been studied yet. One could also dispense with the requirement that the decision tree or tree ensemble classify the training data set perfectly and instead allow a bounded number of examples to be misclassified. This is a better reflection of practical uses of decision trees and tree ensembles. As we observed, Theorem 5.3 can easily be adapted to this setting. If one additionally includes the bound on the number of misclassifed examples as a parameter, the algorithm in Theorem 4.1 can also be adapted by adding a branch in which the dirty example is misclassifed for any dirty example that is considered in the algorithm. Whether or not one can do without this superpolynomial dependence on that bound, may be an interesting question. Also, one could dispense with the restriction that there are only two classes of examples. Theorems 4.1 and 5.3 could likely be adapted, but with a super-polynomial dependence on the number of classes in the running time. An algorithm without this dependence could be a challenge. Finally, a common approach to building tree ensembles is to build them incrementally, with each added tree greedily reducing the number of misclassified examples as much as possible. The parameterized complexity of this problem would be worth investigating as well.

Acknowledgements

Pascal Kunz was supported by the DFG Research Training Group 2434 “Facets of Complexity”. Frank Sommer was supported by the DFG, project EAGR (KO 3669/6-1). Manuel Sorge was supported by the Alexander von Humboldt Foundation.

This work was initiated at the research retreat of the Algorithmics and Computational Complexity group of TU Berlin held in Darlingerode in September 2022.

References

Bessiere, C., Hebrard, E., and O'Sullivan, B. Minimising decision tree size as combinatorial optimisation. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP '09)*, pp. 173–187, 2009. doi: 10.1007/978-3-642-04244-7_16.

Bonamy, M., Kowalik, L., Pilipczuk, M., Socała, A., and Wrochna, M. Tight lower bounds for the complexity of multicoloring. *ACM Transactions on Computation Theory*, 11(3):1–19, 2019. doi: 10.1145/3313906.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001. doi: 10.1023/A:1010933404324.

Carrizosa, E., Molero-Río, C., and Romero Morales, D. Mathematical optimization in classification and regression trees. *TOP*, 29(1):5–33, 2021. doi: 10.1007/s11750-021-00594-1.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. *Parameterized Algorithms*. Springer, 2015. doi: 10.1007/978-3-319-21275-3.

Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., and Stuckey, P. J. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23:1–47, 2022.

Downey, R. G. and Fellows, M. R. *Fundamentals of Parameterized Complexity*. Springer, 2013. doi: 10.1007/978-1-4471-5559-1.

Eiben, E., Ordyniak, S., Paesani, G., and Szeider, S. Learning small decision trees with large domain. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI '23)*. International Joint Conferences on Artificial Intelligence Organization, 2023.

Flum, J. and Grohe, M. *Parameterized Complexity Theory*. Springer, 2006. doi: 10.1007/3-540-29953-X.

Fomin, F. V., Lokshtanov, D., Saurabh, S., and Zehavi, M. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi: 10.1017/9781107415157.

Goodrich, M. T., Mirelli, V., Orletsky, M., and Salowe, J. Decision tree construction in fixed dimensions: Being global is hard but local greed is good, 1995. Technical Report TR-95-1, Department of Computer Science, Johns Hopkins University.

Gottlob, G., Scarcello, F., and Sideri, M. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002. doi: 10.1016/S0004-3702(02)00182-0.

Grigni, M., Mirelli, V., and Papadimitriou, C. H. On the difficulty of designing good classifiers. *SIAM Journal on Computing*, 30(1):318–323, 2000. doi: 10.1137/S009753979630814X.

Hyafil, L. and Rivest, R. L. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976. doi: 10.1016/0020-0190(76)90095-8.

Impagliazzo, R. and Paturi, R. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2): 367–375, 2001. doi: 10.1006/jcss.2000.1727.

Impagliazzo, R., Paturi, R., and Zane, F. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi: 10.1006/jcss.2001.1774.

Kobourov, S. G., Löffler, M., Montecchiani, F., Pilipczuk, M., Rutter, I., Seidel, R., Sorge, M., and Wulms, J. The influence of dimensions on the complexity of computing decision trees. In *Proceedings 37th AAAI Conference on Artificial Intelligence (AAAI '22)*, 2022. Accepted for publication.

Kulkarni, V. Y. and Sinha, D. P. K. Random forest classifiers: A survey and future research directions. *International Journal of Advanced Computing*, 36:1144–1153, 2013.

Narodytska, N., Ignatiev, A., Pereira, F., and Marques-Silva, J. Learning optimal decision trees with sat. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pp. 1362–1368. AAAI Press, 2018. doi: 10.24963/ijcai.2018/189.

Niedermeier, R. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

Ordyniak, S. and Szeider, S. Parameterized complexity of small decision tree learning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pp. 6454–6462, 2021. doi: 10.1609/aaai.v35i7.16800.

Rokach, L. Decision forest: Twenty years of research. *Information Fusion*, 27:111–125, 2016. doi: 10.1016/j.inffus.2015.06.005.

Tamon, C. and Xiang, J. On the boosting pruning problem. In *Proceedings of the 11th European Conference on Machine Learning (ECML '00)*, pp. 404–412, 2000. doi: 10.1007/3-540-45164-1_41.

Verikas, A., Gelzinis, A., and Bacauskiene, M. Mining data with random forests: A survey and results of new tests. *Pattern Recognition*, 44(2):330–349, 2011. doi: 10.1016/j.patcog.2010.08.011.

Vidal, T. and Schiffer, M. Born-again tree ensembles. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, pp. 9743–9753, 2020.