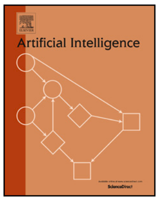
Artificial Intelligence 343 (2025) 104322

Contents lists available at ScienceDirect

Artificial Intelligence

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



The influence of dimensions on the complexity of computing decision trees [☆]

Stephen Kobourov [a], Maarten Löffler [b], Fabrizio Montecchiani [c,*], Marcin Pilipczuk [d], Ignaz Rutter [e], Raimund Seidel [f], Manuel Sorge [g], Jules Wulms [g]

[a] *Technical University Munich, Department of Computer Science, Germany*
[b] *Utrecht University, Department of Information and Computing Sciences, the Netherlands*
[c] *University of Perugia, Department of Engineering, Italy*
[d] *University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics, Poland*
[e] *University of Passau, Faculty of Computer Science and Mathematics, Germany*
[f] *Saarland University, Department of Computer Science, Germany*
[g] *TU Wien, Institute of Logic and Computation, Austria*

ARTICLE INFO

Keywords:
Decision trees
Parameterized complexity
Machine learning

ABSTRACT

A decision tree recursively splits a feature space $\mathbb{R}^d$ and then assigns class labels based on the resulting partition. Decision trees have been part of the basic machine-learning toolkit for decades. A large body of work considers heuristic algorithms that compute a decision tree from training data, usually aiming to minimize in particular the size of the resulting tree. In contrast, little is known about the complexity of the underlying computational problem of computing a minimum-size tree for the given training data. We study this problem with respect to the number d of dimensions of the feature space $\mathbb{R}^d$, which contains n training examples. We show that it can be solved in $O(n^{2d+1})$ time, but under reasonable complexity-theoretic assumptions it is not possible to achieve $f(d) \cdot n^{o(d/\log d)}$ running time. The problem is solvable in $(dR)^{O(dR)} \cdot n^{1+o(1)}$ time if there are exactly two classes and R is an upper bound on the number of tree leaves labeled with the first class.

☆ Main ideas for the results of this paper were developed in the relaxed atmosphere of Dagstuhl Seminar 21062 on *Parameterized Complexity in Graph Drawing*, organized by Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. An extended abstract of this work appeared at AAAI 2023 [18]. In comparison, this version contains full proof details and more discussion. Stephen Kobourov acknowledges funding by the National Science Foundation, grant number NSF-CCF-2212130. Fabrizio Montecchiani acknowledges funding by MUR of Italy, under PRIN Project n. 2022ME9Z78 - NextGRAAL: Next-generation algorithms for constrained GRAph visuALization, and by University of Perugia, Fondi di Ricerca di Ateneo, edizione 2022, project MiRA: Mixed Reality and AI Methodologies for Immersive Robotics. Manuel Sorge acknowledges funding by the Alexander von Humboldt Foundation. Jules Wulms acknowledges funding by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.

* Corresponding author.

E-mail addresses: stephen.kobourov@tum.de (S. Kobourov), m.loffler@uu.nl (M. Löffler), fabrizio.montecchiani@unipg.it (F. Montecchiani), malcin@mimuw.edu.pl (M. Pilipczuk), rutter@fim.uni-passau.de (I. Rutter), rseidel@cs.uni-saarland.de (R. Seidel), manuel.sorge@ac.tuwien.ac.at (M. Sorge), jwulms@ac.tuwien.ac.at (J. Wulms).

https://doi.org/10.1016/j.artint.2025.104322
Received 4 September 2023; Received in revised form 3 March 2025; Accepted 4 March 2025

Available online 7 March 2025

1. Introduction

A decision tree is a useful tool to classify and describe data [24]. It takes the feature space $\mathbb{R}^d$, recursively performs axis-parallel cuts to split the space into two subspaces, and then assigns class labels based on the resulting partition (see Fig. 1). Because of their simplicity, decision trees are particularly attractive as interpretable models of the underlying data [22]. In this context, small trees are preferable, i.e., trees that have a small number of nodes, or in other words, perform a small number of cuts [23]. Such trees are also desired in the context of classification, because it is thought that minimizing the number of nodes reduces the chances of overfitting [8].

In the learning phase, we are given a finite set of examples $E \subseteq \mathbb{R}^d$ labeled with classes and we want to find a decision tree that optimizes certain performance criteria and that is consistent with E, that is, the classes assigned by the tree perfectly agree with the class labels of the examples. Among other criteria, the number of nodes is often minimized for the above-mentioned reasons. There is a plethora of implementations for learning decision trees (e.g., [2,3,14,25,29,30]). The classical CART heuristic herein is among the Top 10 Algorithms of Data Mining chosen by the ICDM [31,32] and several implementations are based on exact algorithms minimizing the size of the produced trees. Despite this, our knowledge of the fine-grained computational complexity of learning (minimum-node) decision trees is limited: Several classical results show NP-hardness [12,15] (see also the survey by Murthy [24]) and we know that, even if we require parameters such as the number of nodes of the tree or the number of different feature values to be small, we still cannot achieve efficient algorithms in terms of upper bounds on the running time [28]. In contrast, if we require to be small the number of nodes of the tree and the largest number of features that distinguish two examples of different classes, then there are prospects for efficient exact algorithms [7].

In this paper, we study the influence of the number d of dimensions of the feature space on the complexity of learning small decision trees. This problem can be phrased as the decision problem MINIMUM DECISION TREE SIZE (DTS): The input is a tuple (E, λ, s) consisting of a set $E \subseteq \mathbb{R}^d$ of examples, a class labeling $\lambda : E \to \{\text{blue}, \text{red}\}$, and an integer s, and we want to decide whether there is a decision tree for (E, λ) of size at most s. Herein, a binary tree T is decision tree for (E, λ) if the labeled partition of $\mathbb{R}^d$ associated with T agrees with the labels λ of E; see Section 2 for a precise definition. By k-DTS we denote the k-class generalization of DTS in which $\lambda : E \to [k]$.

Contributions. We provide the following four main results.

First, we show that DTS can be solved in $O(n^{2d+1}d)$ time, regardless of the number k of label classes, where n is the number of input examples:

Theorem 1. *k-DTS is solvable in $O(D_{\max}^{2d} dn) = O(n^{2d+1}d)$ time.*

Here, $D_{\max}$ denotes the largest feature-domain size, that is, the largest number of values that the input examples can attain in one of the dimensions. Thus, for fixed number d of dimensions, DTS is polynomial-time solvable. Contrast this with the variant where, instead of axis-parallel cuts, we allow linear cuts of arbitrary slopes. This problem is NP-hard already for $d = 3$ [12]. Our algorithm is based on dynamic programming over the relevant possible subhypercubes of the input space. Such hypercubes correspond to potential root-leaf paths in the solution tree. Previously, other researchers used dynamic programming over potential root-leaf paths in the solution as well [27,1,5]. However, they assume that numerical features are binarized, which yields a much worse running time on the order of $O(2^{nd})$ without further modifications.[1]

Second, complementing the first result, we show that the dependency on d in the exponent cannot be substantially reduced. More precisely, a running time of $f(d) \cdot n^{o(d/\log d)}$ would contradict widely accepted complexity-theoretic assumptions:

Theorem 2. *DTS is W[1]-hard with respect to the number d of dimensions. Assuming the Exponential Time Hypothesis there is no algorithm solving DTS in time $f(d)n^{o(d/\log d)}$ where n is the number of input examples and f is a computable function.*

This implies that the running time of algorithms for DTS likely has to scale exponentially with d. In other words, any provably efficient algorithm for DTS has to exploit other properties of the input or desired solution.

Third, we determine more closely what parameters influence the combinatorial explosion, offering two tractability results. A crucial property of a construction that we use in Theorem 2 is that the size of the optimal decision tree is unbounded. Informally, this result thus shows intractability only in situations where the smallest decision tree for our input data is rather large. This may be the case for practical data (partially overlapping classes of Gaussian-distributed data), but it begs the question whether we can find particularly small decision trees provably efficiently if the data allow for it. As Ordyniak and Szeider showed, without further restrictions, this is not possible as DTS is W[2]-hard with respect to the solution size s [28]. In contrast, we show that in the small-dimension regime we do obtain a prospect for an efficient algorithm with running time $O((s^3 d)^s \cdot n^{1+o(1)})$:

Theorem 3. *k-DTS is solvable in $O((s^3 d)^s |E|^{1+o(1)})$ time, and is FPT parameterized by $s + d$.*

[1] It seems not very difficult to modify the extant approaches to yield an improved running time with regard to the number of dimensions. Giving the algorithm directly, however, yields a more concise and self-contained presentation.

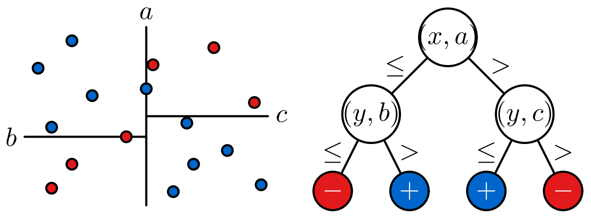


Fig. 1. Left: An instance (E, λ) of DTS with two dimensions x (horizontal) and y (vertical). Points are examples and the blue and red color represents their labels assigned by λ. Right: A minimum decision tree T for (E, λ). Each internal node $t \in T$ is labeled by $(\mathsf{dim}(t), \mathsf{thr}(t))$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

An intermediate result towards Theorem 3 is inspired by and improves upon an algorithm by Ordyniak and Szeider [28] for determining a smallest decision tree that cuts only a given set of features; we decrease the running time from $2^{O(s^2)} \cdot n^{1+o(1)} \cdot \log n$ to $2^{O(s \log s)} \cdot n^{1+o(1)}$ for our purpose:

Corollary 1. *Given a subset $\mathcal{D}$ of dimensions, with $|\mathcal{D}| \le s$, where we are allowed to cut only (a subset of) dimensions in $\mathcal{D}$, DTS is solvable in $2^{O(s \log s)}|E|^{1+o(1)}$ time.*

Since our result appeared, in the meantime Eiben et al. [7] showed that one can replace d by the potentially smaller parameter δ, the largest number of features that distinguish two examples of different classes, maintaining fixed-parameter tractability, that is, a running time of $f(s,\delta) \cdot n^{O(1)}$. In comparison, our dependency of the running time on s, which is exponential in $s \log s$, is better than theirs, which is at least exponential in $s^2 \log s$.

Finally, we show that in the tractability result with respect to s and d, the size s can be replaced by an a priori even smaller parameter: Let R be an upper bound on the number of leaves in the decision tree labeled with any one class. Equivalently, R is an upper bound on the number of parts in the partition induced by the tree that contain only examples of the first class, or that contain only examples of the second class, whichever number is smaller. Then, DTS is solvable in $(dR)^{O(dR)} \cdot n^{1+o(1)}$ time:

Theorem 4. *DTS is solvable in $O((\Delta^3 d)^\Delta |E|^{1+o(1)})$ time, with $\Delta = 2d(2R-1) + R - 1$, and hence DTS is FPT parameterized by $R + d$.*

Theorem 4 follows from a series of observations showing that, in such cases, a minimum-size solution tree has size bounded by $O(dR)$. This result is interesting from a theoretical perspective because DTS is NP-hard even for $R = 1$ in the unbounded-dimension regime [28]. We believe that restricting a decision tree to have a small number R of leaves labeled with one class can also be reasonable in practice: In some cases the distribution of the data may not allow for particularly small decision trees, hampering interpretability. It may then be useful to consider trees in which one class labels few leaves.

Summarizing, while $n^{O(d)}$-time algorithms for DTS are achievable, they likely cannot be substantially improved in general, but when restricting to small solution sizes or restricting a class to have few leaves, there are prospects for efficient algorithms.

Paper outline. Before we give more details on these contributions, we first discuss some preliminaries in Section 2. We then elaborate on our algorithmic results (Theorems 1, 3, and 4) in Section 3, before proving a lower bound on the running time of (k-)DTS (Theorem 2) in Section 4. Section 5 closes out the paper with some concluding remarks.

2. Preliminaries and formal problem definition

For $n \in \mathbb{N}$ we use $[n] := \{1, 2, \dots, n\}$. For a vector $x \in \mathbb{R}^d$ we denote by $x[i]$ the ith entry of x. Let $E \subseteq \mathbb{R}^d$ and $\lambda : E \to \{\text{blue}, \text{red}\}$. Let the domain $D_i := \{x[i] \mid x \in E\}$ of E consist of all distinct coordinate values for dimension i occurring in examples of E. We aim to define what a decision tree for (E, λ) is. Let T be a rooted and ordered binary tree and let $\mathsf{dim} : V(T) \to [d]$ and $\mathsf{thr} : V(T) \to \mathbb{R}$ be labelings of each internal node $t \in V(T)$ by a *dimension* $\mathsf{dim}(t) \in [d]$ and a *threshold* $\mathsf{thr}(t) \in \mathbb{R}$. For each internal node t of T there is a left and a right child of t, labeled by $\le$ and $>$, respectively; see Fig. 1.

We use $E[f_i \le t] = \{x \in E \mid x[i] \le t\}$ and $E[f_i > t] = \{x \in E \mid x[i] > t\}$ to denote the set examples of E whose ith dimension is less or equal and strictly greater than some threshold t, respectively. 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 $E[T,t] := E[T,p] \cap E[f_{\mathsf{dim}(p)} \le \mathsf{thr}(p)]$ 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 $E[t]$.

Now T and the associated labelings are a *decision tree* for (E, λ) if for each leaf ℓ of T we have that all examples in $E[\ell]$ have the same label under λ; we say that such a set $E[\ell]$ is *uniform*. Below we will sometimes omit explicit reference to the labelings of the nodes and edges of T and simply say that T is a decision tree for (E, λ). The *size* of T is the number of its internal nodes. We conveniently call the internal nodes of T and their associated labels *cuts*. The problem Minimum Decision Tree Size (DTS) is defined as in the introduction. For most of our results, the number of classes of the labeling λ does not have to be restricted to two: In most cases our results require checks only on whether two examples have different labels, or on whether a set of examples is

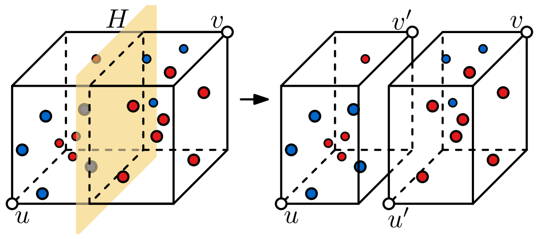


Fig. 2. 3D instance where hyperplane H cuts $\mathrm{box}(u, v)$ into $\mathrm{box}(u, v')$ and $\mathrm{box}(u', v)$, and splits examples $E \cap \mathrm{box}(u, v)$ into $E \cap \mathrm{box}(u, v')$ and $E \cap \mathrm{box}(u', v)$.

uniform. These checks can be agnostic of the number of labels. We therefore also introduce k-DTS as the generalization of DTS in which $\lambda : E \to [k]$. Notice that parameter R in Theorem 4 explicitly assumes $k = 2$, and Theorem 2 already holds for $k = 2$.

Our analysis is within the framework of parameterized complexity [13]. Let $L \subseteq \Sigma^*$ be a computational problem specified over some alphabet Σ and let $p : \Sigma^* \to \mathbb{N}$ 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. A complement to fixed-parameter tractability is W[t]-hardness, $t \geq 1$; if problem L is W[t]-hard with respect to p then it is thought to not be fixed-parameter tractable; see [10,26,4,6] for details. The Exponential Time Hypothesis (ETH) states that 3SAT on n-variable formulas cannot be solved in $2^{o(n)}$ time, see refs. [16,17] for details.

3. Algorithms

We now present our algorithmic results, that is, that k-DTS is polynomial-time solvable for constant number of dimensions, an improved algorithm for the case where we are restricted to a given set of dimensions to cut, a fixed-parameter algorithm for k-DTS with parameter $d + s$ and we show how this algorithm also implies a fixed-parameter algorithm for DTS with parameter $d + R$. For simplicity, we give our algorithms for the decision problem (k-)DTS, but with standard techniques they are easily adaptable to also producing the corresponding tree if it exists and to the corresponding size-minimization problem.

Dynamic programming. For the first result we use the order of the examples E in each dimension. In a dimension i, let $c_1, c_2, \ldots, c_n$ be the coordinate values in D_i in ascending order. We use the set S_i of *splits*, that is, cuts that each partition E in a combinatorially distinct way. Specifically, we can define the set $S_i := \{(c_j + c_{j+1})/2 \mid j \in [n-1]\}$ and another set $S_i^+ := S_i \cup \{-\infty, \infty\}$ with an additional value on either side of the examples. When at most $D_{\max}$ different values are used in each dimension, we find $|S_i| = |D_i| - 1 + 2 \leq D_{\max} + 1 = O(n)$.

Theorem 1. *k-DTS is solvable in $O(D_{\max}^{2d} dn) = O(n^{2d+1} d)$ time.*

Proof. Observe that, by adjusting the thresholds, a minimum decision tree for E can be modified such that for each node t we get $\mathrm{thr}(t) \in S_{\mathrm{dim}(t)}$. Let $S = S_1^+ \times S_2^+ \times \ldots \times S_d^+$.

We do dynamic programming on hyperrectangles defined by two points: for $u, v \in S$, let $\mathrm{box}(u, v)$ be an axis-aligned hyperrectangle with u and v as antipodal corners. We require that each coordinate of u is smaller than the respective coordinate of v. We are interested in computing a minimum decision tree T for the examples in $E \cap \mathrm{box}(u, v)$. This solution can be found by cutting $\mathrm{box}(u, v)$ with an axis-aligned hyperplane H, and combining the minimum-size decision trees T_1, T_2 for the examples on either side of the hyperplane. Since hyperplane H is defined by a dimension i and threshold $t \in S_i$, we can find two new points u' and v', whose coordinates coincide with u and v, respectively, but in dimension i they have coordinate t. Since no example lies on H, by definition of S_i, $E \cap \mathrm{box}(u, v')$ and $E \cap \mathrm{box}(u', v)$ partition the examples in $E \cap \mathrm{box}(u, v)$ (see Fig. 2).

We can therefore use hyperplane H to define the root node (i, t) of decision tree T, with T_1 and T_2 as subtrees. These subtrees are found by recursively computing a solution for $E \cap \mathrm{box}(u, v')$ and $E \cap \mathrm{box}(u', v)$. Since there are i dimensions, and each of them admits at most $D_{\max} + 1$ distinct hyperplanes that we can use to split, an optimal solution for $E \cap \mathrm{box}(u, v)$ can be computed by trying all $O(D_{\max} \cdot d)$ hyperplanes defined by distinct splits in each dimension.

Formally, let $\mathcal{T}$ be a dynamic programming table, in which for each $u, v \in S$ with $u \leq v$, entry $\mathcal{T}[u, v]$ holds the minimum size of a decision tree for $E \cap \mathrm{box}(u, v)$. Define the *volume* of $\mathrm{box}(u, v)$ as the number of grid points contained within it, that is, $|S \cap \mathrm{box}(u, v)|$. We fill the table in a bottom-up fashion from smaller-volume boxes to larger-volume boxes. We iterate over the range of possible volumes ρ from one to $|S|$ and we compute $\mathcal{T}[u, v]$ for all $u, v \in S$ such that $\mathrm{box}(u, v)$ has volume ρ. For each such u, v, we first take $O(nd)$ time to check whether the examples in $\mathrm{box}(u, v)$ all have the same label. If so, then we put $\mathcal{T}[u, v] = 0$. Otherwise, $\mathcal{T}[u, v]$ is defined by the following recurrence:

$$\mathcal{T}[u, v] = \min_{i \in [d]} \min_{\sigma \in S_i} \mathcal{T}[u, v_i(\sigma)] + \mathcal{T}[u_i(\sigma), v] + 1.$$

Thus, for each coordinate i of u and v, we use the values S_i as options. Note that each considered table entry on the right-hand side corresponds to a box of strictly smaller volume than the box on the left-hand side. To see that the recurrence is correct, observe that

Algorithm 1: SMALLESTDECISIONTREE(E, s).

```
Input: Example set $E$ of dimension $d$ and a nonnegative integer $s$
Output: Decision tree $T$ for $E$ with at most $s$ internal nodes using at most $d$ dimensions to label internal nodes.
1  Set sdt to nil, with $|\text{nil}| = \infty$;
2  if $s = 0$ then
3      if $E$ is uniform then return leaf, with $|\text{leaf}| = 0$;
4      else return nil, with $|\text{nil}| = \infty$;
5  for $i = 1$ to $d$ do
6      for $j = 0$ to $s - 1$ do
7          $t = \text{BINARYSEARCH}(E, i, j)$;
8          $r = \text{SMALLESTDECISIONTREE}(E[f_i > t], s - j - 1)$;
9          $l = \text{SMALLESTDECISIONTREE}(E[f_i \le t], j)$;
10         $\text{dt} = (f_i = t) \cup (l, r)$, with $|\text{dt}| = |l| + |r| + 1$;
11         if $|dt| < |sdt|$ then $\text{sdt} = \text{dt}$;
12 return sdt;
```

Algorithm 2: BINARYSEARCH(E, i, j).

```
Input: Example set $E$ of dimension $d$ and nonnegative integers $i$ and $j$
Output: Largest threshold $t$ for which $E[f_i \le t]$ has a decision tree of size $j$
1  Set $D$ to be an array containing $D_i$ in ascending order;
2  Set $L = 0$, $R = |D_i| - 1$, $b = 0$;
3  while $L \le R$ do
4      $m = \lfloor (L + R)/2 \rfloor$;
5      if SMALLESTDECISIONTREE($E[f_i \le D[m]], j$) is not nil then $L = m + 1$, $b = 1$;
6      else $R = m - 1$, $b = 0$;
7  if $b = 1$ then return $D[m]$;
8  return $D[m - 1]$, with $D[-1] = D[0] - 1$
```

a minimum-size decision tree T for $E \cap \text{box}(u, v)$ requires at least one cut. Consider the cut t at the root of T, shifted to a closest split if necessary. At some point while taking the minimum we have $i = \text{dim}(t)$ and $\sigma = \text{thr}(t)$. The subtrees of T rooted at the children of t are decision trees for $E \cap \text{box}(u, v_i(\sigma))$ and $E \cap \text{box}(u_i(\sigma), v)$, respectively. Thus, the left-hand side upper bounds the right-hand side. For the other direction, observe that combining any two decision trees for $E \cap \text{box}(u, v_i(\sigma))$ and $E \cap \text{box}(u_i(\sigma), v)$ with a cut at σ in dimension i gives a decision tree for $E \cap \text{box}(u, v)$, as required. Finally, the size of the minimum-size decision tree for E can be found in $\mathcal{T}[u^*, v^*]$, where all coordinates of u^* and v^* are $-\infty$ and ∞, respectively.

As to the running time, table $\mathcal{T}$ has $O(D_{\max}^{2d})$ entries, each of which each takes $O(nd)$ time to fill: checking for consistency and then trying each distinct split and looking up the size of the respective minimum-size subtrees. We proceed by increasing the domain in each dimension by one split at a time, one coordinate at a time, for a total running time of $O(D_{\max}^{2d} nd) = O(n^{2d+1} d)$. □

With small modifications, 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 an efficiently computable classification error, for example, precision, recall, or F1-score. The idea is to add the size as a parameter to the table $\text{box}(\cdot)$ and instead in $\text{box}(\cdot)$ store, e.g., the minimum number of misclassified examples by any tree of the corresponding size.

Improving on Ordyniak and Szeider [28]. Before we elaborate on our next result, we first show how to improve Theorem 4 in [28], which shows that, given an instance (E, λ, s) of DTS, and given a subset $\mathcal{D}$ of the dimensions, it is possible to compute in $2^{O(s^2)}|E|^{1+o(1)} \log |E|$ time the smallest decision tree among all decision trees of size at most s (if they exist), that use exactly the given subset $\mathcal{D}$ in their cuts — none may be left out. The main idea is to first enumerate the structure of all possible decision trees of size s, before finding thresholds that work for the instance. Instead of enumerating all possible decision trees and finding the right thresholds afterwards, we interleave the two processes, see Algorithm 1. Furthermore, we no longer take as input a subset $\mathcal{D}$ of dimensions, which will be used for labeling internal nodes in the decision tree, but we bound the number d of dimensions in the instance. By iterating over a subset $\mathcal{D}$ instead of all d dimensions (by adding $\mathcal{D}$ as a parameter and modifying Line 5 of Algorithm 1 and Line 1 of Algorithm 2) our approach be adapted towards the initial setting.

Theorem 3. *k-DTS is solvable in $O((s^3 d)^s |E|^{1+o(1)})$ time, and is FPT parameterized by $s + d$.*

Proof. Similar to the algorithm by Ordyniak and Szeider [28], we binary search for the largest threshold value t in dimension i for which the left subtree can still have a decision tree of size j on the example set $E[f_i \le t]$. If we can find a decision tree of size at most $s - j - 1$ for the remaining examples in the right subtree, then we have found a decision tree. However, if we cannot find such a decision tree for the right subtree, then we could not find a decision tree even for smaller threshold values of the root: there would be even more examples left for the right decision tree, which should still be of size at most $s - j - 1$. However, instead of enumerating all trees and assignments of dimension labels to internal nodes, we loop over these options during the main procedure in Algorithm 1.

To see that the recursion in Algorithm 1 leads to a correct outcome, consider the following two cases: In the base case ($s = 0$) the example set E should be uniform, since the resulting decision tree should have $s = 0$ internal nodes. In the step case ($s > 0$) we iterate over all dimensions to decide on the best dimension for the next cut, and we iterate over all sizes for the left subtree (and hence all ways to split $s - 1$ cuts over the two subtrees). As explained in the previous paragraph, the binary search then finds the largest threshold value that still results in a decision tree of size j for the left subtree. Observe that, in case a decision tree of size j is unattainable, the result will be nil, and since $|\mathrm{nil}| = \infty$, this subsolution will never pass the check in Line 11 of Algorithm 1. However, out of all the possible cuts tried in the step case, the one for which the sum of the sizes of the left and right subtrees is minimal, will be stored in sdt in Line 11 and be output. If all cuts lead to a subsolution that is nil, then the output will also be nil, meaning that there is no decision tree of size s for E.

We now prove the running-time bound. Algorithms 1 and 2 together build a recursion tree in which the nodes correspond to calls of Algorithm 1. In each node, corresponding to a call to Algorithm 1 with some set E and numbers s, d, there are at most $sd(2 + \log|E|) = sd\log(4|E|)$ recursive calls to Algorithm 1: For each iteration of the two loops in Algorithm 1 there are two direct recursive calls and at most $\log|E|$ in Algorithm 2. In each recursive call the parameter s decreases by at least one. Hence, the overall size of the recursion tree is $O((sd\log(4|E|))^s)$. For each node N of the recursion tree, we spend $O(|E|)$ time, as the main running time incurred by the call to Algorithm 1 (corresponding to N) is in the uniformity check of E (Line 3 of Algorithm 1); the running time of the remaining bookkeeping tasks in Algorithm 1 and 2 can be charged to the child nodes of N.

Finally, bounding $(\log(4|E|))^s$ uses the following well-known technique: We first prove the claim that $(\log m)^s = O(s^{2s} \cdot m^{1/s})$. To see this, observe that the claim is trivial for $m < s^{2s}$ (as then $(\log m)^s < (2s\log s)^s = s^s(2\log s)^s \le s^{2s}$). Otherwise, if $m \ge s^{2s}$, then we have $\log m \le s^2 m^{1/s^2}$ because this holds for $m = s^{2s}$ (observe that dividing both sides by s yields $2\log s \le s^{1+2/s}$ which clearly holds) and the derivative wrt. m of the left-hand side is at most as large as the derivative (wrt. m) of the right-hand side, that is, $1/(\ln(2)m) < m^{1/s^2-1}$. Thus, in this case $(\log m)^s \le s^{2s}m^{1/s}$, showing the claim. Substituting $4|E|$ for m in $(\log m)^s = O(s^{2s} \cdot m^{1/s})$ we get the overall running time bound of $O((sd)^s \cdot s^{2s} \cdot |E|^{1+1/s}) = O((s^3d)^s \cdot |E|^{1+o(1)})$. □

The strategy employed by Ordyniak and Szeider to solve DTS is to first find a subset $\mathcal{D}$ of dimensions which should be cut to find a smallest decision tree [28]. Once the set $\mathcal{D}$ has been determined, they use their Theorem 4 to find a smallest decision tree. In our case, Algorithm 1 works directly towards the final goal, both selecting dimensions to cut and finding a smallest decision tree at the same time. We can adapt the algorithm to Ordyniak and Szeider's setting of cutting only within a specified set of dimensions by restricting the dimensions to select in Line 5 of Algorithm 1, to obtain a more efficient running time.

Corollary 1. *Given a subset $\mathcal{D}$ of dimensions, with $|\mathcal{D}| \le s$, where we are allowed to cut only (a subset of) dimensions in $\mathcal{D}$, DTS is solvable in $2^{O(s\log s)}|E|^{1+o(1)}$ time.*

A smaller parameter. We now consider the parameter R, the upper bound on the number of leaves in the decision tree labeled with any one class. For simplicity, we will in the following assume that R restricts the number of red leaves in a decision tree, and we assume that there are at least as many blue leaves as red leaves. Observe that this is without loss of generality, because to solve the general case we may simply try both options. Below, we call an internal node that has red leaves in both subtrees an *essential* node. We will show that in a minimum-size tree there is a bounded number of both essential and non-essential nodes. Afterwards, applying the algorithm from Theorem 3 will give us a fixed-parameter algorithm for $d + R$.

Lemma 1. *A minimum-size decision tree T with R red leaves has at most $R - 1$ essential nodes.*

Proof. Consider the subtree T', whose root is the essential node closest to the root of T. There is only one such node, as either the root of T is essential, or one of its subtrees contains only blue leaves. Observe that T' is a binary tree, by virtue of being a decision tree, and hence only its root is a degree-2 node; all other internal nodes have degree 3, and leaves have degree 1. Remove from T' all nodes that have only blue leaves as descendants or are blue leaves themselves. New degree-2 nodes in the resulting tree, introduced by this procedure, must be non-essential: they are internal nodes for which one child or one subtree is removed. Contracting these degree-2 nodes, we again obtain a binary tree T^*. Note that the internal nodes of T^* one-to-one correspond to the essential nodes of T. Tree T^* still has all R red leaves of T (and no blue ones), and thus consists of at most $R - 1$ internal nodes. □

Lemma 2. *In a minimum-size decision tree T with R red leaves, each root-to-leaf path has at most $2d$ consecutive non-essential nodes, where d is the number of dimensions.*

Proof. Assume for a contradiction that a minimum-size decision tree T exists for example set E that has $2d + 1$ consecutive non-essential nodes on a root-to-leaf path p. Let n_0 be the essential node that is the child of the $2d + 1$-th consecutive non-essential node in p. Since there are $2d + 1$ non-essential nodes, there are three nodes $n_1 = (i, t_1)$, $n_2 = (j, t_2)$, and $n_3 = (l, t_3)$ such that $i = j = l$. Additionally, at least two of those nodes have a blue leaf as a child on the same side (left or right). Assume w.l.o.g. that n_1 and n_2 have a blue leaf as their right child, $t_1 < t_2$, and either $t_3 < t_1$ or $t_2 < t_3$. Thus, n_1 is closer to n_0 than n_2 on the path p (see Fig. 3).

Consider a decision tree T^* that is identical to T, except it does not contain n_2, nor the blue leaf n_4 attached at n_2 (as its right child) — the parent of n_2 is directly connected to the internal node that is the left child of n_2. The node n_0^* in T^*, corresponding to n_0 in T, is unaffected by this change, meaning that $E[T, n_0] = E[T^*, n_0^*]$, for the following reason. All blue examples in $E[T, n_4]$ which

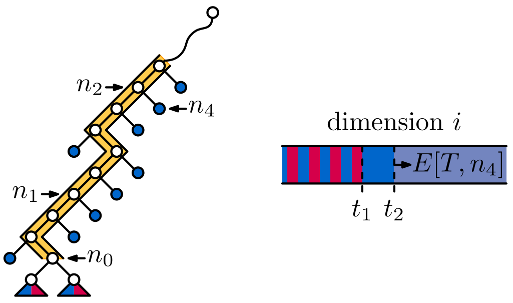


Fig. 3. Construction of Lemma 2: path p in yellow, and nodes n_1 and n_2 with thresholds t_1 and t_2 in dimension i.

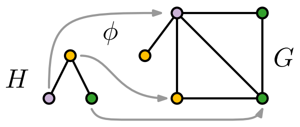


Fig. 4. A subgraph isomorphism ϕ (in gray) from the vertices of H to the vertices of G.

follow the root-to-leaf path to n_1^* in T^* (corresponding to n_1 in T), will not reach n_0^*, since in dimension $i = j$ each such example e has a coordinate c_e, for which holds that $t_1 < t_2 < c_e$. Thus, such examples belong in the leaf node connected to n_1^*. As a result, T^* is a smaller decision tree for E than T, contradicting the assumption that T has minimum size. □

Lemmas 1 and 2 together show that a minimum size decision tree has at most $2d$ non-essential nodes before each essential node and each red leaf, and hence has at most $R - 1$ essential internal nodes and at most $2d(2R - 1)$ non-essential internal nodes. We can therefore apply Theorem 3 to prove that DTS is FPT with d and R as parameters.

Theorem 4. *DTS is solvable in $O((\Delta^3 d)^\Delta |E|^{1+o(1)})$ time, with $\Delta = 2d(2R - 1) + R - 1$, and hence DTS is FPT parameterized by $R + d$.*

4. Running-time lower bound

We now prove a lower bound for computing decision trees.

Theorem 2. *DTS is W[1]-hard with respect to the number d of dimensions. Assuming the Exponential Time Hypothesis there is no algorithm solving DTS in time $f(d)n^{o(d/\log d)}$ where n is the number of input examples and f is a computable function.*

The remainder of this section is devoted to the proof of Theorem 2. Below, for a graph G we use $V(G)$ to denote its vertex set and $E(G)$ for its edge set. We give a reduction from the Partitioned Subgraph Isomorphism (PSI) problem. Its input consists of a graph G with a proper K-coloring $\mathsf{col}: V(G) \to [K]$, that is, no two adjacent vertices share a color, and a graph H with vertex set $[K]$ that has no isolated vertices. The question is whether H is isomorphic to a subgraph of G that respects the colors, i.e., whether there exists a mapping $\phi: V(H) \to V(G)$ such that (i) each vertex of H is mapped to a vertex of G with its color, i.e., $\mathsf{col}(\phi(c)) = c$ for each $c \in V(H)$ and (ii) for each edge $\{c, c'\}$ in H, $\{\phi(c), \phi(c')\} \in E(G)$ is an edge of G. See Fig. 4 for an example. We say that ϕ is a *subgraph isomorphism* from H into G.

In the following, we let $m_G = |E(G)|$, $n_H = |V(H)|$ and $m_H = |E(H)|$. Observe that $n_H \le 2m_H$ since H has no isolated vertices. For each color $k \in [K]$, we denote by $V_k = \{v \in V(G) \mid \mathsf{col}(v) = k\}$ the vertices of G with color k. We assume without loss of generality that there is an $i \in \mathbb{N}$ such that for all $k \in [K]$ we have $|V_k| = i$ (otherwise add additional isolated vertices to G as needed) and that if there is no edge in H between two vertices $u, v \in V(H)$, then there are no edges between V_u and V_v in G.

Since PSI contains the Multicolored Clique problem [9] as a special case, PSI is W[1]-hard with respect to m_H. Moreover, Marx [21, Corollary 6.3] observed that an $f(m_H) \cdot n^{o(m_H/\log m_H)}$-time algorithm for PSI would contradict the Exponential Time Hypothesis. Our reduction will transfer this property to DTS parameterized by the number of dimensions.

Proof outline. Given an instance (G, H) of PSI our goal is to construct an equivalent instance (E, λ) of DTS. To solve the instance of PSI, we can select edges from the graph G; one edge for each edge in H. Specifically, we consider sets of edges in G for which the endpoints have colors c and c' corresponding an edge $\{c, c'\}$ in H. We select one edge from each such set and verify that these edges in G actually form a graph that is isomorphic with H. This is done by considering every selected edge in G with an endpoint of color c, and checking that all such endpoints correspond to one and the same vertex in G. This must be the case for a subgraph isomorphism, as there is only one vertex c in H (for each color c). In our reduction, we emulate this strategy by constructing appropriate selection and verification gadgets.

Before we elaborate on our construction, let us first also consider our options for the instance (E, λ) of DTS that we are going to construct. We have to construct a set E of examples in a d-dimensional space, and provide a class labeling λ that assigns one of two

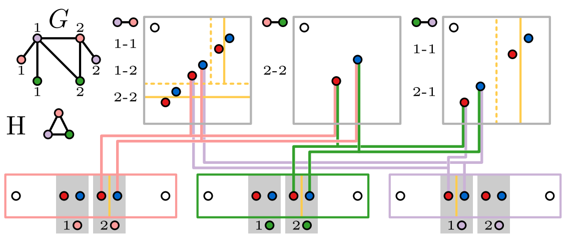


Fig. 5. An instance (E, λ) for graphs G and H. Gray squares show edge-selection subspaces. Edges of H and indices of colored vertices in G are shown left of each subspace. Colored rectangles show vertex-verification subspaces for the corresponding colors. Unlabeled points are white. We (visually) connect points in different subspaces to show the projection of examples in those subspaces. The yellow cuts form a solution to DTS corresponding to a solution to PSI for (G, H); dashed cuts indicate omitted dummy tuples.

color (red or blue) to each example. Note that the latter restriction ensures that the reduction also works for k-DTS, with $k > 2$, by simply using less than k colors. For our construction we work towards an instance that is of polynomial size compared to the instance (G, H) of PSI. Additionally, we will ensure that the number d of dimensions for the examples in E is linear in the number m_H of edges of H in order to transfer parameterized hardness from PSI with respect to m_H to hardness for DTS with respect to d. Furthermore, our goal is to ensure that a set of orthogonal cuts separating the red from the blue examples in our constructed instance, corresponds to selecting edges from G and verifying that the edges and their endpoints form a subgraph isomorphism from H to G.

To model the edges and vertices of graph G, the selection of edges, and verification of vertices for the subgraph isomorphism from H to G, we order the examples E differently in different dimensions. That is, we reserve certain dimensions for specific purposes, such as 2-dimensional spaces for edge selection and single dimensions for vertex verification. See Fig. 5 for an overview of how these concepts relate — more details will be filled in later. The crux of our reduction is that we must ensure that the cuts separating the examples act largely independently: In certain dimensions cuts determine which edges are chosen, and in other dimensions cuts verify that the endpoints of the chosen edges coincide. These cuts must therefore interact in very precise and restricted ways. For example, only when a subgraph isomorphism is possible with the selected edges, should a low number of cuts separate all red examples from all blue examples. Additionally, placing more cuts to select edges should not make it possible to place less cuts that correspond to verifying the endpoints, and vice versa. If this would be possible, then we lose the correspondence between our constructed instance of DTS and the instance of PSI we are reducing from.

To ensure that such a correspondence exists, we use pairs of examples that correspond to edges or vertices of G and H; the *primary examples*. Additionally we choose the placement of the examples in the d-dimensional space, such that they have different coordinates only in certain dimensions. Only if a red and a blue example pair are on different coordinates in a certain dimension, can a cut in that dimension split those examples. This insight gives us some way of bringing structure to the sets of cuts that separate all red examples from blue examples. We put certain pairs of red and blue examples on different coordinates only in one or a few dimensions, to force obligatory cuts in those dimensions. To structure the solution in our constructed instance even further, we introduce what we call *dummy examples*; pairs of examples that do not directly correspond to vertices or edges of G or H, but that simply exist to force cuts.

Construction. Following the outline above, our construction consists of two types of gadgets. First, for each edge $\{c, c'\} \in E(H)$, we use a two-dimensional *edge-selection subspace* to model the choice for an edge $\{u, v\} \in E(G)$ with $\mathrm{col}(u) = c, \mathrm{col}(v) = c'$. Second, for each vertex $c \in V(H)$, we use a one-dimensional *vertex-verification subspace* to check whether the chosen edges with an endpoint of color c consistently end in the same vertex $u \in V(G)$ with $\mathrm{col}(u) = c$. We use primary examples to model vertices and edges in G, while dummy examples are used only to force certain cuts in the constructed instance.

We first describe the constructed instance (E, λ) of DTS by giving point sets for the edge-selection and vertex-verification subspaces. Later, we define the examples in E by choosing the points they project to in each of the subspaces. Besides the exception mentioned below, these points will be labeled and primary examples may project only to points with a matching label (red or blue), whereas dummy examples can project to any point. In each subspace there will be several points that will be used by multiple examples in order to achieve the correct behavior of each gadget. On the other hand, most examples will play a role only in very few subspaces and the other dimensions shall not be relevant for them. To achieve this property, we reserve in each subspace one unlabeled point (usually with the minimum or the maximum coordinate) that can be used by all examples that shall not be separated from each other in this specific subspace. The vertex-verification subspaces have a second unlabeled point that can only be used by dummy examples. We call an example that projects to an unlabeled point of some subspace *irrelevant* for this subspace and, conversely, the subspace is irrelevant for this example.

We need some more tools to describe the points in the edge-selection and vertex-verification subspaces: In one-dimensional subspaces, the precise coordinates of the points do not matter and we rather specify their order. The main ingredient in the construction are pairs of a red and a blue point that need to be separated: An *rb-pair* is a pair (r, b) of points that are consecutive in the linear order with the red point preceding the blue point. As explained before, to give structure to the possible solutions to our constructed instance, we separate the *rb*-pairs with forced cuts. To achieve this, we use what we call dummy tuples. A *dummy tuple* consists of $2(m_G + 2)$ points (*dummy points*) to which only dummy examples can project — in the upcoming paragraphs we will elaborate on why this high number of dummy points is helpful. The first two are red, the second two are blue, and so on, and the last two are blue (without loss of generality, we assume that m_G is even); see Fig. 6. Dummy tuples are placed between consecutive *rb*-pairs. We later project dummy examples to the points in a dummy tuple so to ensure the following two properties. First, only examples

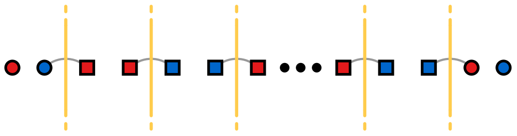


Fig. 6. Two rb-pairs (disks) and dummy tuples (squares) between them, which force the yellow cuts. Examples projecting to connected points differ only in this dimension.

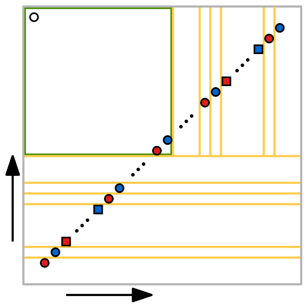


Fig. 7. An edge-selection subspace. Consecutive red-blue pairs are shown as disks; dummy examples are squares. The unlabeled point is white. The yellow cuts show how one pair will be left unseparated, and cuts can be placed such that this pair is not separated from the unlabeled point either.

with labels matching the respective point in the tuple can project to such a point. Second, these examples force $m_G + 3$ cuts in the corresponding subspace as follows. A number of $m_G + 1$ cuts must be placed between each pair of equally labeled and adjacent middle dummy points, since we ensure that the examples that project here differ only in the subspace of this dummy tuple and in no other subspace. Additionally, two cuts must be placed between the outer dummy points and the adjacent rb-pair, since we ensure that the dummy examples that project to the outer points differ from an example projecting to the neighboring point in the rb-pair only in the subspace of the dummy tuple.

Edge-selection. We now describe a two-dimensional edge-selection subspace S_e for an edge e of H, see Fig. 7 for an illustration. We refer to the two dimensions of S_e by *x and y-dimension of* S_e. Except for the unlabeled point, each point p has coordinates of the form (c_p, c_p), and we therefore simply specify the linear point order. Let $e_1, e_2, \ldots, e_j$ denote the edges of G whose endpoints have the same colors as e. For each edge e_i, we place an rb-pair called *e_i's edge pair*. Between any two edge pairs we put a dummy tuple. We place an unlabeled point whose x-coordinate is smaller than that of any other point and whose y-coordinate is larger than that of any other point. We allocate a budget of $(j-1) \cdot (m_G + 4)$ cuts that shall be used for performing cuts in these two dimensions. The idea is that, by using $(j-1) \cdot (m_G + 3)$ cuts, it is possible to have a cut between every edge pair and the dummy tuples adjacent to it ($2(j-1)$ cuts) and the inner pairs of each dummy tuple ($(j-1) \cdot (m_G + 1)$ cuts), and the remaining $j - 1$ cuts can then be used to cut all but one of the edge pairs, which corresponds to choosing the edge whose edge pair is not cut. Conversely, for each edge e of H, there is a decision tree of size $(j-1) \cdot (m_G + 4)$ for the points in the subspace, for which only the example set of a single leaf contains both red and blue points, and it contains precisely the edge pair of the edge e_i of G and the unlabeled point; see Fig. 7.

The crucial property of the dummy tuples that we will leverage is that each tuple contributes at least $m_G + 1$ cuts. We will place dummy examples at these dummy points in such a way that we ensure the strict interaction between edge-selection and vertex verification we foreshadowed in the outline: if a solution tries to leave multiple edge pairs in an edge-selection subspace unseparated, then an additional $m_G + 1$ cuts (on top of the initial $m_G + 1$) will be required for each dummy tuple between the edge pairs (of which there is at least one). As we will explain in more detail later, if this happens in even a single edge-selection subspace, the number of cuts will instantly exceed the upper bound s on the number of cuts that are required to find a subgraph isomorphism.

Vertex-verification. Next, we describe a vertex-verification subspace for a vertex v of H. The vertex-verification subspace of v is one-dimensional, and we again describe its projection by giving the order of the labeled points. We first place a left unlabeled point, then one rb-pair, called a *vertex pair*, for each vertex of G whose color is v, and then a right unlabeled point; see Fig. 5. We allocate a budget of a single cut for each vertex-verification space. This budget allows us to place a cut that separates one vertex pair as well as the left unlabeled and the right unlabeled point. The idea is that all but the vertex pair that corresponds to the vertex of G that has been selected shall be separated by cuts in the edge-selection subspaces, and therefore a single cut suffices in the vertex-verification subspace.

Synthesis and examples. We now describe the instance (E, λ, s) of DTS that we construct for a given instance (G, H) of PARTITIONED SUBGRAPH ISOMORPHISM. Our examples are elements of a space that contains m_H two-dimensional edge-selection subspaces and n_H one-dimensional vertex-verification subspaces, i.e., our points are in dimension $d = 2m_H + n_H$. According to the budgets of cuts for the subspaces given above, we put the upper bound s on the size of the desired decision tree to be $s = (m_G + 4) \cdot (m_G - m_H) + n_H$.

Our construction contains two red and two blue primary examples for each edge of G. For each edge $e = \{u, v\}$ of G, let S_{uv} denote the edge-selection subspace corresponding to the edge $\{\mathsf{col}(u), \mathsf{col}(v)\}$ of H and let S_u and S_v denote the vertex-verification subspaces corresponding to $\mathsf{col}(u)$ and $\mathsf{col}(v)$, respectively. We create two primary example pairs U and V, each consisting of a red and a blue example, which project to the vertex pair corresponding to u and v in S_u and in S_v, respectively. They both project to the edge pair of e in S_{uv}. In all other dimensions, these pairs project to the (right) unlabeled point.

We now describe the dummy examples. We create for each dummy tuple D contained in an edge-selection subspace S a number of $2(m_G + 1) + 1$ pairs of examples $L_1, L_2, \ldots, L_{m_G+1}, R_1, R_2, \ldots, R_{m_G+1}, P$ that each consist of a red and a blue dummy example. In subspace S, each pair L_i and R_i both project to the ith pair of adjacent red and blue points in the middle of D. In all other edge-selection subspaces, they project to the unlabeled point. In each vertex-verification subspace, the pairs L_i and R_i project to the left and the right unlabeled point, respectively — this is a crucial part of the construction on which we further elaborate in the next paragraph. The red example of the pair P projects in S to the outer red point of D, and it coincides in all other subspaces with some fixed blue primary example b that projects to the blue point preceding it in S. Likewise, the blue example of P projects in S to the outer blue point of D, and it coincides in all other subspaces with some fixed red primary example r that projects to the red point succeeding it in S. Observe that examples of P can be separated from r and b only in subspace S, and therefore force the presence of two cuts. Similarly, each L_i and R_i and the remaining examples are separable only in S.

Finally, we create for each vertex-verification subspace S one pair U of a red and blue dummy example, which project to the left and to the right unlabeled point in S, respectively. In all other dimensions, they project to the (right) unlabeled point. A key technicality is that pair U enforces at least one cut in S. However, if we separate U by some cut C before separating the pairs L_i and R_i of some dummy tuple D, then L_i and R_i end up on different sides of C, increasing the necessary cuts in the edge-selection subspace of D.

Lemma 3. *Instance (G, H, col) of PSI is a yes-instance, if and only if instance (E, λ, s) of DTS is a yes-instance.*

Proof. $\Rightarrow$: If (G, H, col) is a yes-instance for PSI, then there is a subgraph isomorphism ϕ from H to G. We construct a decision tree for (E, λ, s) as follows. The decision tree's inner nodes form a path and thus we specify the sequence of cuts that we make in the nodes of the path. We start by placing cuts in the edge-selection subspace S_e for each edge $e \in E(H)$. Denote the edges of G whose endpoints match the colors of e by $e_1, e_2, \ldots, e_j$ as in the description of S_e. Let i be such that $\phi(e) = e_i$. We first make the following cuts in the y-dimension of S_e in ascending order. Cut between the edge pairs of $e_1, e_2, \ldots, e_{i-1}$, and make all cuts prescribed by dummy tuples succeeding $e_1, \ldots, e_{i-1}$. Then we make the following cuts in the x-dimension of S_e in descending order. Cut between the edge pairs of $e_j, e_{j-1}, \ldots, e_{i+1}$, and make all cuts prescribed by dummy tuples preceding $e_j, e_{j-1}, \ldots, e_{i+1}$; see Fig. 7. Note that these are $(m_G + 4) \cdot (j - 1)$ cuts in this subspace, and in total $(m_G + 4) \cdot (m_G - m_H)$ cuts in all of the m_H edge-selection subspaces. Additionally, note that each cut has one side on which there are only red or only blue examples, i.e., these cuts correspond to a decision tree for a subset of the examples and the tree's internal nodes form a path. After performing these cuts for all edges of H, all pairs of dummy examples associated with dummy tuples are separated from each other. The only primary example pairs that are not yet separated are those that correspond to the edges whose edge pairs have not been cut in the edge-selection subspaces. These examples are not separated from each other either, because they are not separated from the irrelevant examples (projected to the unlabeled point) in their respective subspaces.

Afterwards, for each vertex c of H, we cut between the vertex pair of the vertex $\phi(c) \in V(G)$ in the vertex-verification subspace S of c. Observe that this separates both the dummy example D of S and one of the two pairs of examples that corresponds to the selected edges incident to $\phi(c)$ (the other one is separated by the corresponding cut in the vertex-verification subspace of the other endpoint; see Fig. 5). Note that each of these n_H cuts separates a set of red examples from the remaining examples, and they can be performed in an arbitrary order. In total, we have used $(m_G + 4) \cdot (m_G - m_H) + n_H = s$ cuts, to separate the red from the blue examples, and in fact, the internal nodes of the decision tree form a path.

$\Leftarrow$: Now assume that (E, λ) admits a decision tree T with s cuts. First, observe that for each dummy tuple, the tree T contains $m_G + 3$ distinct cuts by the way we have defined the projections of the dummy examples. In total, there are $(m_G + 3) \cdot (m_G - m_H)$ such cuts. Furthermore, since in each vertex-verification subspace there is a dummy pair, there are further n_H distinct cuts in T. Call all of these cuts *required*.

Say that an edge of G is *chosen* if its primary red and blue examples are not cut apart in an edge-selection subspace by T. Among all solution decision trees, pick T such that it has the minimum number of nodes that make cuts in vertex-verification subspaces. We claim that, for each edge e of H, there is at most one chosen edge in G whose endpoints have the same colors as e. For a contradiction, assume that edges f and f' of G are chosen and their endpoints' colors correspond to the endpoints of $e \in E(H)$. Since the primary examples of f and f' only differ in S_e and in vertex-verification subspaces, these examples are cut apart in vertex-verification subspaces. Let t, t' denote nodes in T that cut the primary examples of f and f', respectively, via a cut in a vertex-verification subspace. Consider the lowest common ancestor a of t and t'. Observe that $E[a]$ contains both the primary examples of f and f'. Thus, $E[a]$ also contains all examples that occur in the dummy tuples between f and f' in the edge-selection subspace S_e.

Consider one such dummy tuple D and let b, b' be the children of a. Consider the case where a equals t or t', that is, a is a cut in a vertex-verification subspace corresponding to the endpoints of e. Then, a cuts each R_i of D from each L_i of D. Thus for each $i \in \{1, 2, \ldots, m_G + 1\}$ it is impossible for T to cut the pair R_i and L_i at the same time in the subtree of T rooted at a and thus, in addition to the required cuts, there are $m_G + 1$ further cuts in T. That is, the number of cuts in T is at least $(m_G + 3) \cdot (m_G - m_H) + n_H + m_G + 1 > (m_G + 4) \cdot (m_G - m_H) + n_H = s$, a contradiction. Thus, a is not equal to t or t'. If a does not cut at least one primary example of f from at least one primary example of f', then t and t' both reside in a subtree rooted at b or in a subtree rooted at b', contradicting that a is the lowest common ancestor of t and t'. Thus, assume that a cuts at least one primary example of f from at least one primary example of f'. Again, these primary examples only differ in S_e and in vertex-verification subspaces. This implies that a is either a cut in a vertex-verification subspace corresponding to an endpoint of e or a cut in S_e somewhere between the primary examples of

f and f'. The first case, we already considered, namely a is equal to t or t', which leads to a contradiction. So assume that a is a cut in S_e somewhere between the primary examples of f and f'. Since the unlabeled point in S_e is on exactly one side of this cut and all examples that are irrelevant for S_e project to the unlabeled point, it follows that either $E[b]$ or $E[b']$ contains only dummy examples or primary examples of edges with the same colors as e; say $E[b]$ does. Assume that t is in the subtree of T rooted at b. This is without loss of generality by renaming. Note that $E[t]$ cannot contain two pairs of primary examples corresponding to different edges: In this case, $E[t]$ contains also all examples of the dummy tuples between the two edges in S_e. Performing the cut at t in a vertex-verification subspace would hence introduce $m_G + 1$ cuts in addition to the required ones (as before), a contradiction. Thus, $E[t]$ contains among primary examples only those of f. Hence, we may replace t by a cut in S_e, a contradiction, because T has the minimum number of nodes that make cuts in vertex-verification subspaces. Thus, indeed, for each edge e of H, there is at most one chosen edge in G whose endpoints have the same colors as e.

By the calculation of required cuts it now follows that also there is at least one chosen edge for each edge e of H. For each edge e in H the primary examples of the chosen edge for e need to be separated in both vertex-verification subspaces corresponding to the endpoints of e. Furthermore, for each vertex-verification subspace there is one required cut in that subspace. Thus, two chosen edges whose endpoints have the same color are incident with the same vertex of that color. Thus, the chosen edges induce a subgraph isomorphism ϕ from H to G. □

As the reduction takes polynomial-time, and $d = 2m_H + n_H \leq 4m_H$, Theorem 2 readily follows.

5. Conclusion

We have begun charting the tractability for learning small decision trees with respect to the number d of dimensions. While exponents in the running time need to depend on d, this dependency is captured by the size of the desired tree or the number of leaves labeled with the fewest leaves. It would be interesting to analyze what other features can capture the combinatorial explosion induced by dimensionality; this can be done by deconstructing our hardness result [19]. Parameters that are necessarily unbounded for the reduction to work include the number of examples that have the same feature values and the maximum number of alternations between labels when sorting examples in a dimension.

Recently, researchers started investigating exact algorithms for computing optimal tree ensembles [20], that is, computing sets of decision trees of minimum total size whose majority vote classifies the training data correctly. It would be interesting to see how our results extend to computing optimal tree ensembles. Our $n^{o(d/\log d)}$-time lower bound should be easy to adapt, but it is not clear whether the same $O(n^{2d+1})$-time upper bound is achievable.

We have focused on computing decision trees that classify the training examples fully correctly. In practice, often one would allow for small margins of error. For straightforward formulations of such weaker accuracy guarantees, this results in a more general problem formulation that encompasses Minimum Decision Tree Size as a special case. Thus, our running-time lower bound directly applies also to such more general problems. In the positive direction, the algorithm from Theorem 1 can easily be adapted to computing trees that implement Pareto-optimal trade-offs between size and measures of accuracy. For our other algorithmic results, we leave such extensions as open problems. In this regard, it was recently shown [11] that some of the parameterized algorithms [28,7] can only be extended if one adds the number of outliers as a parameter. This would potentially preclude efficient algorithms for the case for a large number of input examples when we allow for a constant fraction of misclassified examples. It would be interesting to see if tractability results in the small-dimension regime are more robust in this regard.

Another interesting direction is the following. Counter to our theoretical result, it may be the case that in practice sometimes indeed training with data on larger number of dimensions can be done more efficiently, because potentially such high-dimensional data is easier to separate. Clearly, training data with smaller number of dimensions can be embedded into training data with larger number of dimensions and thus in general the computational complexity only increases with the number of dimensions. In order to prove potential efficiency gains with larger number of dimensions one would thus have to capture mathematically the right properties of data that allow easier separation with growing number of dimensions. It would be interesting to know what such properties might be and whether they occur in practice.

CRediT authorship contribution statement

Stephen Kobourov: Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Maarten Löffler:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Fabrizio Montecchiani:** Writing – review & editing, Writing – original draft, Investigation, Conceptualization. **Marcin Pilipczuk:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Ignaz Rutter:** Writing – review & editing, Writing – original draft, Validation, Conceptualization. **Raimund Seidel:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Manuel Sorge:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization. **Jules Wulms:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Fabrizio Montecchiani reports financial support was provided by Government of Italy Ministry of Education University and

Research. Stephen Kobourov reports financial support was provided by National Science Foundation. Manuel Sorge reports financial support was provided by Alexander von Humboldt Foundation. Jules Wulms reports financial support was provided by Vienna Science and Technology Fund.

Acknowledgements

We thank the anonymous reviewers of this paper for their valuable comments and suggestions.

Data availability

No data was used for the research described in the article.

References

[1] G. Aglin, S. Nijssen, P. Schaus, Learning optimal decision trees using caching branch-and-bound search, in: Proc. 34. AAAI Conference on Artificial Intelligence, AAAI 2020, ISBN 978-1-57735-823-7, 2020, pp. 3146–3153.
[2] C. Bessiere, E. Hebrard, B. O'Sullivan, Minimising decision tree size as combinatorial optimisation, in: Proc. 15th International Conference on Principles and Practice of Constraint Programming, CP, 2009, pp. 173–187.
[3] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Chapman & Hall/CRC, 1984.
[4] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, 2015.
[5] E. Demirovic, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, P.J. Stuckey, MurTree: optimal decision trees via dynamic programming and search, J. Mach. Learn. Res. 23 (2022) 26.
[6] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer, 2013.
[7] E. Eiben, S. Ordyniak, G. Paesani, S. Szeider, Learning small decision trees with large domain, in: Proc. 32nd International Joint Conference on Artificial Intelligence, IJCAI 2023, 2023, pp. 3184–3192, ijcai.org.
[8] U.M. Fayyad, K.B. Irani, What should be minimized in a decision tree?, in: Proc. 8th National Conference on Artificial Intelligence, AAAI, 1990, pp. 749–754, https://www.aaai.org/Library/AAAI/1990/aaai90-112.php.
[9] M.R. Fellows, D. Hermelin, F. Rosamond, S. Vialette, On the parameterized complexity of multiple-interval graph problems, Theor. Comput. Sci. 410 (1) (2009) 53–61, https://doi.org/10.1016/j.tcs.2008.09.065.
[10] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer, 2006.
[11] H. Gahlawat, M. Zehavi, Learning small decision trees with few outliers: a parameterized perspective, in: M.J. Wooldridge, J.G. Dy, S. Natarajan (Eds.), Proc 38. AAAI Conference on Artificial Intelligence, AAAI 2024, AAAI Press, 2024, pp. 12100–12108.
[12] M.T. Goodrich, V. Mirelli, M. Orletsky, J. Salowe, Decision tree construction in fixed dimensions: being global is hard but local greed is good, Technical Report TR-95-1, Department of Computer Science, Johns Hopkins University, May 1995.
[13] G. Gottlob, F. Scarcello, M. Sideri, Fixed-parameter complexity in AI and nonmonotonic reasoning, Artif. Intell. 138 (1–2) (2002) 55–86, https://doi.org/10.1016/S0004-3702(02)00182-0.
[14] X. Hu, C. Rudin, M.I. Seltzer, Optimal sparse decision trees, in: Proc. 33rd Annual Conference on Neural Information Processing Systems, NeurIPS, 2019, pp. 7265–7273, https://proceedings.neurips.cc/paper/2019/hash/ac52c626afc10d4075708ac4c778ddfc-Abstract.html.
[15] L. Hyafil, R.L. Rivest, Constructing optimal binary decision trees is NP-complete, Inf. Process. Lett. 5 (1) (1976) 15–17, https://doi.org/10.1016/0020-0190(76)90095-8.
[16] R. Impagliazzo, R. Paturi, On the complexity of k-SAT, J. Comput. Syst. Sci. 62 (2) (2001) 367–375, https://doi.org/10.1006/jcss.2000.1727.
[17] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, J. Comput. Syst. Sci. 63 (4) (2001) 512–530, https://doi.org/10.1006/jcss.2001.1774.
[18] S.G. Kobourov, M. Löffler, F. Montecchiani, M. Pilipczuk, I. Rutter, R. Seidel, M. Sorge, J. Wulms, The influence of dimensions on the complexity of computing decision trees, in: Proc. 37th AAAI Conference on Artificial Intelligence, AAAI 2023, AAAI Press, 2023, pp. 8343–8350, https://ojs.aaai.org/index.php/AAAI/article/view/26006.
[19] C. Komusiewicz, R. Niedermeier, J. Uhlmann, Deconstructing intractability—a multivariate complexity analysis of interval constrained coloring, J. Discret. Algorithms 9 (1) (2011) 137–151, https://doi.org/10.1016/j.jda.2010.07.003.
[20] C. Komusiewicz, P. Kunz, F. Sommer, M. Sorge, On computing optimal tree ensembles, in: Proc. 39th International Conference on Machine Learning, ICML 2023, vol. 202, PMLR, 2023, pp. 17364–17374, https://proceedings.mlr.press/v202/komusiewicz23a.html.
[21] D. Marx, Can you beat treewidth?, Theory Comput. 6 (1) (2010) 85–112, https://doi.org/10.4086/toc.2010.v006a005.
[22] C. Molnar, Interpretable machine learning, Independently published, https://christophm.github.io/interpretable-ml-book, 2020.
[23] M. Moshkovitz, S. Dasgupta, C. Rashtchian, N. Frost, Explainable k-means and k-medians clustering, in: Proc. 37th International Conference on Machine Learning, ICML, 2020, pp. 7055–7065, http://proceedings.mlr.press/v119/moshkovitz20a.html.
[24] S.K. Murthy, Automatic construction of decision trees from data: a multi-disciplinary survey, Data Min. Knowl. Discov. 2 (4) (1998) 345–389, https://doi.org/10.1023/A:1009744630224.
[25] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva, Learning optimal decision trees with SAT, in: Proc. 27th International Joint Conference on Artificial Intelligence, IJCAIs, 2018, pp. 1362–1368.
[26] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford, 2006.
[27] S. Nijssen, É. Fromont, Optimal constraint-based decision tree induction from itemset lattices, Data Min. Knowl. Discov. 21 (1) (2010) 9–51, https://doi.org/10.1007/S10618-010-0174-X.
[28] S. Ordyniak, S. Szeider, Parameterized complexity of small decision tree learning, in: Proc. 35th AAAI Conference on Artificial Intelligence, AAAIs, 2021, pp. 6454–6462, https://ojs.aaai.org/index.php/AAAI/article/view/16800.
[29] A. Schidler, S. Szeider, Sat-based decision tree learning for large data sets, J. Artif. Intell. Res. 80 (2024) 875–918, https://doi.org/10.1613/JAIR.1.15956.
[30] L.P. Staus, C. Komusiewicz, F. Sommer, M. Sorge Witty, An efficient solver for computing minimum-size decision trees, in: Proc 39. AAAI Conference on Artificial Intelligence, AAAI 2025, AAAI Press, 2025, in press, https://arxiv.org/abs/2412.11954.
[31] D. Steinberg, CART: classification and regression trees, in: X. Wu, V. Kumar (Eds.), The Top Ten Algorithms in Data Mining, Chapman & Hall/CRC, 2009, pp. 179–201.
[32] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z.-H. Zhou, M. Steinbach, D.J. Hand, D. Steinberg, Top 10 algorithms in data mining, Knowl. Inf. Syst. 14 (1) (2008) 1–37, https://doi.org/10.1007/s10115-007-0114-2.