The Influence of Dimensions on the Complexity of Computing Decision Trees

Stephen G. Kobourov[1], Maarten Löffler[2], Fabrizio Montecchiani[3], Marcin Pilipczuk[4], Ignaz Rutter[5], Raimund Seidel[6], Manuel Sorge[7], Jules Wulms[7]

[1] University of Arizona, Department of Computer Science
[2] Utrecht University, Department of Information and Computing Sciences
[3] University of Perugia, Department of Engineering
[4] University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics
[5] University of Passau, Faculty of Computer Science and Mathematics
[6] Saarland University, Department of Computer Science
[7] TU Wien, Institute of Logic and Computation
kobourov@cs.arizona.edu, m.loffler@uu.nl, fabrizio.montecchiani@unipg.it, malcin@mimuw.edu.pl, rutter@fim.uni-passau.de, rseidel@cs.uni-saarland.de {manuel.sorge, jwulms}@ac.tuwien.ac.at

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.

1 Introduction

A decision tree is a useful tool to classify and describe data (Murthy 1998). 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 Figure 1). Because of their simplicity, decision trees are particularly attractive as interpretable models of the underlying data (Molnar 2020). 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 (Moshkovitz et al. 2020). 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 (Fayyad and Irani 1990).

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., (Breiman et al. 1984; Bessiere, Hebrard, and O'Sullivan 2009; Narodytska et al. 2018; Schidler and Szeider 2021; Hu, Rudin, and Seltzer 2019)). The classical CART heuristic herein is among the Top 10 Algorithms of Data Mining chosen by the ICDM (Wu et al. 2008; Steinberg 2009) and several implementations are based on exact algorithms minimizing the size of the produced trees. Despite this, our knowledge of the computational complexity of learning (minimum-node) decision trees is limited: Several classical results show NP-hardness (Hyafil and Rivest 1976; Goodrich et al. 1995) (see also the survey by Murthy (1998)) 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 (Ordyniak and Szeider 2021).

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 $\lambda\colon E \to \{\text{blue}, \text{red}\}$, 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.

We provide three main results. First, we show that DTS can be solved in $O(n^{2d+1}d)$ time (Theorem 3.1), where n is the number of input examples. In other words, for fixed number 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 $d = 3$ (Goodrich et al. 1995).

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 4.1). This implies that the running time

of algorithms for DTS 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, a pair of results that determines more closely what parameters influence the combinatorial explosion, offering two tractability results. A crucial property of a construction that we use in Theorem 4.1 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 (Ordyniak and Szeider 2021). In contrast, we show that in the small-dimension regime we do obtain a prospect for an efficient algorithm with running time $O((s^3d)^s \cdot n^{1+o(1)})$ (Theorem 3.2). An intermediate result towards this is inspired by and improves upon an algorithm by Ordyniak and Szeider (2021) 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.

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 3.6). This is interesting from a theoretical perspective because DTS is NP-hard even for $R = 1$ in the unbounded-dimension regime (Ordyniak and Szeider 2021). 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 cannot be substantially improved in general, but when restricting to small solution sizes or a class to have few leaves, there are prospects for efficient algorithms.

2 Preliminaries

For $n \in \mathbb{N}$ we use $[n] := \{1, 2, \ldots, n\}$. 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 $\lambda \colon E \to \{\text{blue}, \text{red}\}$. 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 $\mathsf{dim} \colon V(T) \to [d]$ 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* $\mathsf{thr}(t) \in \mathbb{R}$. For each internal node t of T there is a left and a right child of t, labeled by $\leq$ and $>$, respectively; see Figure 1.

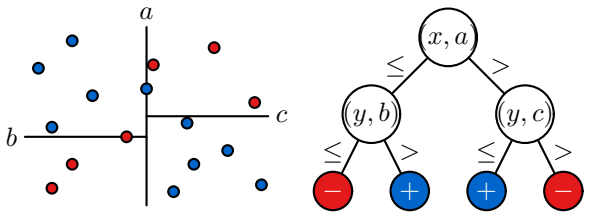

Figure 1: Left: An instance (E, λ) of DTS with two dimensions x (horizontal) and y (vertical). Points are examples and the blue and red color represents their labels assigned by λ. Right: A minimum decision tree T for (E, λ). Each internal node $t \in T$ is labeled by $(\mathsf{dim}(t), \mathsf{thr}(t))$.

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 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 $E[T, t] := E[T, p] \cap E[f_{\mathsf{dim}(p)} \leq \mathsf{thr}(p)]$ 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 $E[t]$.

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. 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. We therefore also introduce k-DTS as the generalization of DTS in which $\lambda \colon E \to [k]$.

Our analysis is within the framework of parameterized complexity (Gottlob, Scarcello, and Sideri 2002). Let $L \subseteq \Sigma^*$ be a computational problem specified over some alphabet Σ and let $p \colon \Sigma^* \to \mathbb{N}$ 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 $f(p) \cdot \text{poly}(n)$ 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 (Flum and Grohe 2006; Niedermeier 2006; Cygan et al. 2015; Downey and Fellows 2013) for details. The Exponential Time Hypothesis (ETH) states that 3SAT on n-variable formulas cannot be solved in $2^{o(n)}$ time, see refs. (Impagliazzo and Paturi 2001; Impagliazzo, Paturi, and Zane 2001) for details.

3 Algorithms

We now present our algorithmic results, that is, that k-DTS is polynomial-time solvable for constant number of

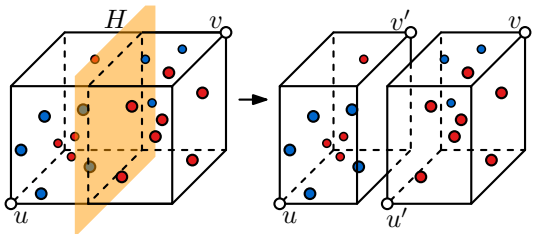

Figure 2: 3D instance where hyperplane H cuts $\mathrm{box}(u, v)$ into $\mathrm{box}(u, v')$ and $\mathrm{box}(u', v)$, and splits examples $E \cap \mathrm{box}(u, v)$ into $E \cap \mathrm{box}(u, v')$ and $E \cap \mathrm{box}(u', v)$.

dimensions, an improved algorithm for the case where we are restricted to a given set of dimensions to cut, a fixed-parameter algorithm for $d + s$ and one for $d + R$. 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.

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 $S_i := \{(c_j + c_{j+1})/2 \mid j \in [n-1]\} \cup \{-\infty, \infty\}$. When at most $D_{\max}$ different values are used in each dimension, we find the bound: $|S_i| = |D_i| - 1 + 2 \leq D_{\max} + 1 = O(n)$.

Theorem 3.1. *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 $\mathsf{thr}(t) \in S_{\mathsf{dim}(t)}$. 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 Figure 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 a minimum-size decision tree T for $E \cap \mathrm{box}(u, v)$ 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 $i = \mathsf{dim}(t)$ and $\sigma = \mathsf{thr}(t)$. The subtrees of T rooted at the children of t are decision trees for $E \cap \mathrm{box}(u, v_i(\sigma))$ and $E \cap \mathrm{box}(u_i(\sigma), v)$, respectively. Thus, the left-hand side upper bounds the right-hand side. For the other direction, observe that combining any two decision trees for $E \cap \mathrm{box}(u, v_i(\sigma))$ and $E \cap \mathrm{box}(u_i(\sigma), v)$ with a cut at σ in dimension i gives a decision tree for $E \cap \mathrm{box}(u, v)$, 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. Thus, the total running time adds up to $O(D_{\max}^{2d} nd) = O(n^{2d+1} d)$. □

Before we elaborate on our next result, we first show how to improve Theorem 4 in (Ordyniak and Szeider 2021), 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 exists), 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 simply bound the number d of dimensions in the instance. By iterating over a subset $\mathcal{D}$ instead of all d dimensions, Algorithm 1 can be adapted towards the initial setting.

Theorem 3.2. *k-DTS is solvable in $O((s^3 d)^s |E|^{1+o(1)})$ time, and is FPT parameterized by $s + d$.*

Algorithm 1: SMALLESTDECISIONTREE(E, d, s)

Input: Example set E and numbers s and d
Output: Decision tree T for E with at most s internal nodes using d dimensions to label internal nodes.

1 Set sdt to nil, with $|\mathsf{nil}| = \infty$;
2 **if** $s = 0$ **then**
3 **if** E *is uniform* **then return** leaf, with $|\mathsf{leaf}| = 0$;
4 **else return** nil, with $|\mathsf{nil}| = \infty$;
5 **for** $i = 1$ *to* d **do**
6 **for** $j = 0$ *to* $s - 1$ **do**
7 $t = \text{BINARYSEARCH}(E, i, j)$;
8 $r = \text{SMALLESTDECISIONTREE}(E[f_i > t], d, s - j - 1)$;
9 $l = \text{SMALLESTDECISIONTREE}(E[f_i \leq t], d, j)$;
10 dt $= (f_i = t) \cup (l, r)$, with $|\mathsf{dt}| = |l| + |r| + 1$;
11 **if** $|dt| < |sdt|$ **then** sdt $=$ dt;
12 **return** sdt;

Algorithm 2: BINARYSEARCH(E, i, j)

Input: Example set E and numbers i and j
Output: Largest threshold t for which $E[f_i \leq t]$ 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** $L \leq R$ **do**
4 $m = \lfloor (L + R)/2 \rfloor$;
5 **if** SMALLESTDECISIONTREE$(E[f_i \leq D[m]], d, j)$ *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 $D[-1] = D[0] - 1$

Proof sketch. Similar to the algorithm by Ordyniak and Szeider (Ordyniak and Szeider 2021), 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 $E[f_i \leq t]$. 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.

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 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 $(\log m)^s < (2s \log s)^s = s^s (2 \log s)^s \leq s^{2s}$). Otherwise, if $m \geq s^{2s}$, then we have $\log m \leq s^2 m^{1/s^2}$ because this holds for $m = s^{2s}$ (observe that dividing both sides by s yields $2 \log s \leq s^{1+2/s}$ 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 $(\log m)^s \leq s^{2s} m^{1/s}$, 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 $O((sd)^s \cdot s^{2s} \cdot |E|^{1+1/s}) = O((s^3 d)^s \cdot |E|^{1+o(1)})$. □

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 (Ordyniak and Szeider 2021). 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 3.3. *Given a subset $\mathcal{D}$ of dimensions, with $|\mathcal{D}| \leq s$, 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.*

We now consider the parameter R. 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.

Lemma 3.4. *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. Remove from T' all nodes that have only blue leaves as descendants or are blue leaves themselves. The resulting tree contains degree-2 nodes, which

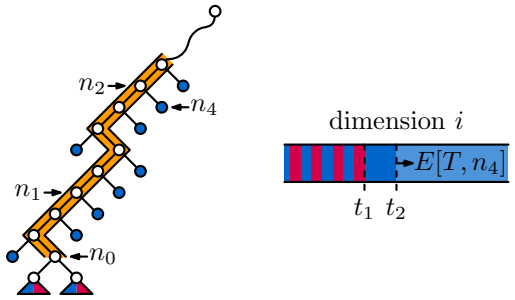

Figure 3: Construction of Lemma 3.5: path p in yellow, and nodes n_1 and n_2 with thresholds t_1 and t_2 in dimension i.

must be non-essential. Contracting the 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 3.5. *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 their child. 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 Figure 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 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 3.4 and 3.5 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 3.2 to prove that DTS is FPT with d and R as parameters.

Theorem 3.6. DTS *is solvable in $O((s^3 d)^s |E|^{1+o(1)})$ time, with $s = 2d(2R-1) + R - 1$, and hence* DTS *is FPT parameterized by $d + R$.*

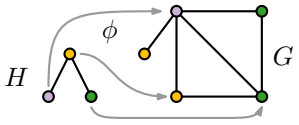

Figure 4: A subgraph isomorphism ϕ (in gray) is a mapping from the vertices of H to vertices of G.

4 Running-Time Lower Bound

We now prove a lower bound for computing decision trees.

Theorem 4.1. MINIMUM DECISION TREE SIZE (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 input size and f is a computable function.*

The remainder of this section is devoted to the proof of Theorem 4.1. 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 $\mathsf{col}\colon V(G) \to [K]$, 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 $\phi\colon V(H) \to V(G)$ 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 Figure 4 for an example of such a mapping. In that case, we also 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 $n_H \leq 2m_H$ 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 $n \in \mathbb{N}$ such that for all $k \in [K]$ we have $|V_k| = n$ (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 (Fellows et al. 2009) as a special case, PSI is W[1]-hard with respect to m_H. Moreover, Marx (Marx 2010, 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.

Outline. Given an instance (G, H) of PSI we now describe how to construct an equivalent instance (E, λ) of DTS. 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 $\mathsf{col}(u) = c, \mathsf{col}(v) = c'$. 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 $\mathsf{col}(u) = c$. Furthermore, we classify examples in our construction into two types: *pri-*

mary and *dummy* examples. 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 the labeled point sets that we obtain when projecting the examples in E to the edge-selection and vertex-verification subspaces. Later, we define the examples in E by giving the points they project to in each of the subspaces. We specify labels for most of these points, 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 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 vertices. We call an example that projects to the 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. To avoid that rb-pairs interfere with each other, we separate them 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. 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 Figure 5. 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 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 Figure 6 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.

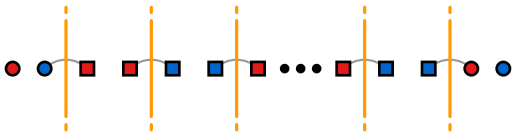

Figure 5: Two rb-pairs (disks) and dummy tuples (squares) between them, which force the yellow cuts. Examples projecting to connected points differ only in this dimension.

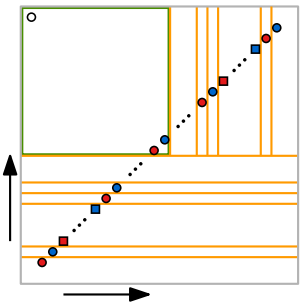

Figure 6: An edge-selection subspace. Consecutive red-blue pairs are shown as disks; dummy examples are squares. The unlabeled point is white. The yellow cuts show how one pair will be left unseparated, and cuts can be placed such that this pair is not separated from the unlabeled point either.

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 Figure 6.

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 Figure 7. We allocate a budget of a single cut for each vertex-verification space. This budget allows 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 project to a 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. 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 dummy pair U whose red and blue examples 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 both of the pairs L and R of some dummy tuple D, then L and R end up on different sides of C, increasing the necessary cuts in the edge-selection subspace of D.

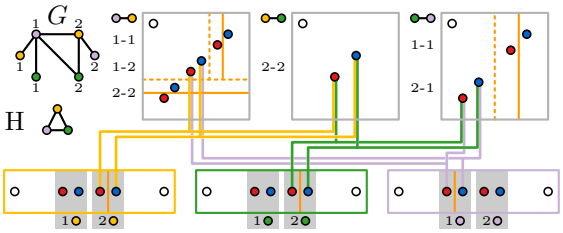

Figure 7: An instance (E, λ, s) for graphs G and H. Gray squares show edge-selection subspaces. Edges of H and indices of colored vertices in G are shown left of each subspace. Colored rectangles show vertex-verification subspaces for the corresponding colors. Unlabeled points are white. We connect points 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, col); dashed cuts indicate omitted dummy tuples.

Lemma 4.2. *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 sketch. $\Rightarrow$: 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. First place $(m_G + 4) \cdot (n - 1)$ cuts in each edge-selection subspace S_e for each edge $e \in E(H)$. Only one pair of primary examples is left unseparated (see Figure 6), for an edge e_i of G, with $\phi(e) = e_i$. We place $(m_G + 4) \cdot (m_G - m_H)$ cuts in total.

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. 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 in the subspace of the other endpoint; see Figure 7). In total we get $(m_G + 4) \cdot (m_G - m_H) + n_H = s$ cuts to separate the red from the blue examples. $\Leftarrow$: Now assume that (E, λ) admits a decision tree T with s cuts. Observe that $(m_G + 3) \cdot (m_G - m_H) + n_H$ cuts between dummy examples are always required. We argue that in each edge-selection subspace at most one primary pair can be left unseparated and those edges induce a subgraph isomorphism from H to G (Kobourov et al. 2022). □

Since the reduction takes polynomial-time, and $d = 2m_H + n_H \leq 4m_H$, Theorem 4.1 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 number of leaves labeled with the first class, the class with the fewest leaves. It would be interesting to analyse what other features can capture the combinatorial explosion induced by dimensionality; this can be done by deconstructing our hardness result (Komusiewicz, Niedermeier, and Uhlmann 2011). 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.

Acknowledgements

Main ideas for the results of this paper were developed in the relaxed atmosphere of Dagstuhl Seminar 21062 on *Pa-*

rameterized Complexity in Graph Drawing, organized by Robert Ganian, Fabrizio Montecchiani, Martin Nöllenburg, and Meirav Zehavi. Stephen Kobourov acknowledges funding by the National Science Foundation, grant number NSF-CCF-2212130. Fabrizio Montecchiani acknowledges funding by University of Perugia, Fondi di Ricerca di Ateneo, edizione 2021, project "AIDMIX - Artificial Intelligence for Decision making: Methods for Interpretability and eXplainability". 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.

References

Bessiere, C.; Hebrard, E.; and O'Sullivan, B. 2009. Minimising Decision Tree Size as Combinatorial Optimisation. In *Proc. 15th International Conference on Principles and Practice of Constraint Programming (CP)*, 173–187.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Chapman & Hall/CRC.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

Downey, R. G.; and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Fayyad, U. M.; and Irani, K. B. 1990. What Should Be Minimized in a Decision Tree? In *Proc. 8th National Conference on Artificial Intelligence (AAAI)*, 749–754.

Fellows, M. R.; Hermelin, D.; Rosamond, F.; and Vialette, S. 2009. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1): 53–61.

Flum, J.; and Grohe, M. 2006. *Parameterized Complexity Theory*. Springer.

Goodrich, M. T.; Mirelli, V.; Orletsky, M.; and Salowe, J. 1995. 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.

Gottlob, G.; Scarcello, F.; and Sideri, M. 2002. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell.*, 138(1-2): 55–86.

Hu, X.; Rudin, C.; and Seltzer, M. I. 2019. Optimal Sparse Decision Trees. In *Proc. 33th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 7265–7273.

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

Impagliazzo, R.; and Paturi, R. 2001. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2): 367–375.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4): 512–530.

Kobourov, S. G.; Löffler, M.; Montecchiani, F.; Pilipczuk, M.; Rutter, I.; Seidel, R.; Sorge, M.; and Wulms, J. 2022. The Influence of Dimensions on the Complexity of Computing Decision Trees. *CoRR*, abs/2205.07756.

Komusiewicz, C.; Niedermeier, R.; and Uhlmann, J. 2011. Deconstructing intractability—A multivariate complexity analysis of interval constrained coloring. *Journal of Discrete Algorithms*, 9(1): 137–151.

Marx, D. 2010. Can You Beat Treewidth? *Theory of Computing*, 6(1): 85–112.

Molnar, C. 2020. *Interpretable Machine Learning*. Independently published.

Moshkovitz, M.; Dasgupta, S.; Rashtchian, C.; and Frost, N. 2020. Explainable k-Means and k-Medians Clustering. In *Proc. 37th International Conference on Machine Learning (ICML)*, 7055–7065.

Murthy, S. K. 1998. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4): 345–389.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning optimal decision trees with SAT. In *Proc. 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 1362–1368.

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

Ordyniak, S.; and Szeider, S. 2021. Parameterized Complexity of Small Decision Tree Learning. In *Proc. 35th AAAI Conference on Artificial Intelligence (AAAI)*, 6454–6462.

Schidler, A.; and Szeider, S. 2021. SAT-based Decision Tree Learning for Large Data Sets. In *Proc. 35th AAAI Conference on Artificial Intelligence (AAAI)*, 3904–3912.

Steinberg, D. 2009. CART: Classification and Regression Trees. In Wu, X.; and Kumar, V., eds., *The Top Ten Algorithms in Data Mining*, 179–201. Chapman & Hall/CRC.

Wu, X.; Kumar, V.; Ross Quinlan, J.; Ghosh, J.; Yang, Q.; Motoda, H.; McLachlan, G. J.; Ng, A.; Liu, B.; Yu, P. S.; Zhou, Z.-H.; Steinbach, M.; Hand, D. J.; and Steinberg, D. 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1): 1–37.