Properly learning decision trees with queries is NP-hard

Caleb Koch

Stanford

Carmen Strassle Stanford

Li-Yang Tan Stanford

July 11, 2023

Abstract

We prove that it is NP-hard to properly PAC learn decision trees with queries, resolving a longstanding open problem in learning theory (Bshouty 1993; Guijarro–Lavín–Raghavan 1999; Mehta–Raghavan 2002; Feldman 2016). While there has been a long line of work, dating back to (Pitt–Valiant 1988), establishing the hardness of properly learning decision trees from *random examples*, the more challenging setting of *query* learners necessitates different techniques and there were no previous lower bounds. En route to our main result, we simplify and strengthen the best known lower bounds for a different problem of Decision Tree Minimization (Zantema–Bodlaender 2000; Sieling 2003).

On a technical level, we introduce the notion of *hardness distillation*, which we study for decision tree complexity but can be considered for any complexity measure: for a function that requires large decision trees, we give a general method for identifying a small set of inputs that is responsible for its complexity. Our technique even rules out query learners that are allowed constant error. This contrasts with existing lower bounds for the setting of random examples which only hold for inverse-polynomial error.

Our result, taken together with a recent almost-polynomial time query algorithm for properly learning decision trees under the uniform distribution (Blanc–Lange–Qiao–Tan 2022), demonstrates the dramatic impact of distributional assumptions on the problem.

Contents

1	Introduction 1			
	1.1 Background and Context	2		
	1.2 Other related work	2		
	1.3 Technical remarks about Theorem 1	3		
2	Technical Overview	4		
	2.1 Why the query setting necessitates new techniques	4		
	2.2 Overview of our proof and techniques	5		
	2.2.1 The core reduction \ldots	5		
	2.2.2 Hardness distillation	6		
	2.2.3 Hardness for constant error	7		
3	Discussion and future work			
4	Preliminaries	9		
	4.1 Hardness of Vertex Cover	11		
5	A reduction from VertexCover to Decision Tree Minimization	11		
	5.1 Intuition and warmup: the IsEdge _G function \ldots	11		
	5.1.1 Useful notions and notation: edge partitions and divergent path prefixes	12		
	5.1.2 Proof of Claim 2.1	13		
	5.2 ℓ -IsEdge: an amplified version of IsEdge	15		
	5.2.1 Proof of Theorem 2 \ldots	17		
	5.3 Hardness of decision tree minimization	19		
6	Hardness distillation and learning consequence for small error	20		
	6.1 A general method for hardness distillation	20		
	6.2 Warmup: hardness distillation for IsEdge	23		
	6.3 Hardness distillation for ℓ -IsEdge	25		
	6.4 Learning consequence for inverse polynomial error	28		
7	Hardness for constant error	30		
	7.1 Hardness of partial vertex cover	30		
	7.2 Definition of the hard distribution	31		
	7.3 Learning consequence for constant-error: Proof of Theorem 1	34		

1 Introduction

Decision trees are among the most basic and popular hypothesis classes in machine learning. They have long served as the gold standard of interpretability: a classic, influential survey of statistical models states that "On interpretability, decision trees rate an A+" [Bre01], and two decades later, a survey of interpretable machine learning [RCC⁺22] lists the optimization of decision tree hypotheses as the very first of the field's "10 grand challenges". Besides interpretability, decision tree hypotheses are extremely fast to evaluate, with evaluation time scaling with their depth, a quantity that is often exponentially smaller than their overall size. Decision trees are also at the heart of powerful ensemble methods such as random forests and XGBoost which achieve state-of-the-art performance across a variety of domains.

We consider the task of constructing optimal decision tree representations of data. A standard formalization of this task is the problem of properly PAC learning decision trees: given access to a target function f and a distribution \mathcal{D} , construct the optimal decision tree hypothesis for funder \mathcal{D} . Valiant's original definition of the PAC model [Val84] considered learners with both passive access to the target function in the form of random labeled examples as well as active access in the form of queries. This setting as well as that of learning from random examples only have since been intensively studied. Valiant's motivation for the more powerful query setting came from modeling interactions with an expert ("[an] important aspect of the formulation is that the notion of oracles makes it possible to discuss a whole range of teacher-learner interactions beyond the mere identification of examples"). The query setting also models the task of converting an existing accurate but inscrutable hypothesis, for which one has query access to, into a more intelligible representation—once again, decision trees are a canonical sought-for representation for this task [CS95, BS96, VAB07, ZH16, BKB17, VLJ⁺17, FH17, VS20].

This work. The NP-hardness of properly learning decision trees from random examples is a foundational result known since the early days of PAC learning [Ang, PV88]. The question of whether there exists an efficient query learner, on the other hand, has been raised repeatedly over the years, in research papers [Bsh93, GLR99, MR02] and surveys [Fel16]. We resolve this question by showing that properly learning decision trees is NP-hard even for query learners:

Theorem 1. There is an absolute constant $\varepsilon > 0$ such that the following holds. Suppose there is an algorithm that, given queries to an n-variable function f computable by a decision tree of size s = O(n) and random examples $(\boldsymbol{x}, f(\boldsymbol{x}))$ drawn according to a distribution \mathcal{D} , runs in time t(n) and w.h.p. outputs a size-s decision tree h that is ε -close to f under \mathcal{D} . Then SAT \in RTIME(poly($t(n^2 \operatorname{polylog} n)))$).

Theorem 1 addresses a stark gap in our understanding of the problem. The fastest known algorithm runs in exponential time, $2^{O(n)}$ for all values of s. There were no previous lower bounds, leaving open the possibility of a poly(n, s)-time algorithm. Indeed, existing query learners for various relaxations of the problem had suggested that such an algorithm was within striking distance. Theorem 1 provides for the first time strong evidence that there are no polynomial-time, or indeed even subexponential-time, algorithms for the problem.

1.1 Background and Context

Hardness of properly learning decision trees from random examples. NP-hardness in the setting of random examples has been known since the seminal work of Pitt and Valiant [PV88]. Their paper, which initiated the study of the hardness of proper learning, attributed the result to an unpublished manuscript of Angluin [Ang]. Subsequently, Hancock, Jiang, Li, and Tromp [HJLT96] obtained hardness even of *weakly-proper* learning, where the algorithm is allowed to return a decision tree of size larger than that of the target. There have since been several works [ABF⁺09, KST23, Bsh23] further improving [HJLT96]'s result.

These works build crucially on a simple reduction from SETCOVER, a reduction variously attributed to Levin [Lev73], Angluin [Ang], and Haussler [Hau88]. We describe this technique and discuss why it is limited to the setting of random examples in Section 2.1.

Algorithms for properly learning decision trees. There is a simple Occam algorithm for properly learning decision trees from random examples: for a size-s decision tree target, draw $O(s \log(n))$ many labeled examples and use dynamic programming to find a size-s decision tree hypothesis that fits the dataset perfectly (see e.g. [GLR99, MR02]). Standard generalization bounds [BEHW89] show that this algorithm satisfies the PAC guarantee. Its runtime is $2^{O(n)}$, with the dynamic program being the bottleneck.

Ehrenfeucht and Haussler [EH89] gave a faster algorithm that runs in time $n^{O(\log s)}$, but their algorithm is only weakly proper: for a size-s target, its hypothesis can be as large as $n^{\Omega(\log s)}$. This large gap is a significant drawback—decision tree hypotheses are interpretable and fast to evaluate insofar as they are *small*—and [EH89] stated as the first open problem of their paper that of designing algorithms that produce smaller hypotheses. There has been no progress on this problem in the setting of random examples since 1989.

The power of queries. In contrast, granting the learner queries enables the design of several polynomial-time algorithms that almost solve the problem of properly learning decision trees. Already in his original paper [Val84] (see also [Ang88]), Valiant gave a polynomial-time query algorithm for properly learning monotone DNFs; consequently, for size-s monotone decision tree targets Valiant's algorithm returns a size-s monotone DNF as its hypothesis. Other such results include polynomial-time query learners for general decision tree targets that output depth-3 formulas [Bsh93] and polynomials [KM93, SS93] as hypotheses. As further demonstration of the power of queries, a recent work of Blanc, Lange, Qiao, and Tan [BLQT22] gives an almost-polynomial-time (poly(n) · $s^{O(\log \log s)}$ time) query algorithm that properly learns decision trees under the uniform distribution. Finally, the query model opens the possibility of circumventing long-known SQ lower bounds for the problem [BFJ⁺94], which show that in the setting of random examples all SQ algorithms must take time $n^{\Omega(\log s)}$.

Taken together, this was all evidence in favor of a polynomial-time, or at least a mildly-superpolynomial time algorithm for properly learning decision trees with queries. In light of Theorem 1, even a subexponential-time algorithm is now unlikely.

1.2 Other related work

Scarcity of hardness results for PAC learning with queries. Theorem 1 adds to a dearth of NP-hardness results for the model of PAC learning with queries. Indeed, we are aware of only

one other such result: in [Fel06] Feldman proved that DNFs are NP-hard to properly learn with queries, resolving a longstanding problem of Valiant [Val84, Val85]. As Feldman remarked in his paper, this was the first NP-hardness result, for any learning task, for the model of PAC learning with queries. (Our techniques are entirely different from [Fel06]'s.)

Related to the scarcity of hardness results, there are numerous query algorithms, for a variety of learning tasks, whose runtimes remain unmatched in the setting of random examples. It is also well known that under standard cryptographic assumptions, PAC learning with queries is strictly more powerful than from random examples only.

Hardness of properly learning decision trees in other models. While the focus of our work is on the PAC model, interest in the hardness of properly learning decision trees predates and extends beyond the PAC model. An early paper by Hyafil and Rivest [HR76] proved the NP-hardness of constructing generalized decision trees (ones with more expressive splits than the values of single variables) that perfectly fit a given dataset; quoting the authors, "The importance of this result can be measured in terms of the large amount of effort that has been put into finding efficient algorithms for constructing optimal binary decision trees". Other results on the hardness of properly learning decision trees in other models include [GJ79, KPB99, GLR99, ZB00, BB03, LN04, CPR⁺07, RRV07, Sie08, AH12, Rav13, BLQT21].

1.3 Technical remarks about Theorem 1

Hardness for constant error. A notable aspect of Theorem 1 is that it rules out learners that are allowed constant error. This was not known even in the setting of random examples, where existing hardness results only hold for inverse-polynomial error: prior to our work, there were no lower bounds ruling out algorithms for properly learning size-s DTs, from random examples only, in time say $(ns)^{O(1/\varepsilon)}$, which is polynomial for constant ε . (Feldman's NP-hardness result for properly learning DNFs with queries also only holds for inverse-polynomial error.)

Implications for decision tree minimization. The actual result that we prove is stronger than as stated in Theorem 1: it holds even if the learner is given explicit descriptions of the target function f and the distribution \mathcal{D} as inputs. Furthermore, the target function can even be given to the learner in the form of a decision tree. For this reason, our result also has implications for the problem of decision tree minimization: given a decision tree, find an equivalent one of minimum size. We recover the best known hardness of approximation result for this problem [ZB00, Sie08] via what is, in our opinion, a much simpler proof. Our proof also yields a stronger result: we show that the problem remains hard even if the resulting tree only has to agree with the original tree on a small given set of inputs.

Implications for testing decision trees. Another aspect in which the actual result we prove is stronger than as stated in Theorem 1 is that it even rules out distribution-free *testers* for decision trees. (The fact that lower bounds against testers for a class yield lower bounds against proper learners for the same class is well known and easy to show [GGR98].) While there's a large body of work giving lower bounds for testing various classes of functions, the vast majority of these results are information-theoretic in nature, focusing on query complexity, with far fewer computational lower bounds. Our result does not rule out decision tree testers with low query complexity, but it shows that even if such a tester exists, it must nevertheless run in exponential time (unless SAT admits a subexponential time algorithm).

2 Technical Overview

2.1 Why the query setting necessitates new techniques

Before delving into our techniques, we describe the key construction [Lev73, Ang, Hau88] at the heart of all previous results on the hardness of properly learning decision trees from random examples [Ang, HJLT96, ABF⁺09, KST23, Bsh23] and discuss why it is limited this setting. (This section can be freely skipped; its point is to explain why we had to depart from previous approaches in order to prove Theorem 1.)

Consider the following reduction from SETCOVER to the problem of properly learning disjunctions. Let $S = \{S_1, \ldots, S_n\}$ be a SETCOVER instance over the universe [m] and define $u^{(1)}, \ldots, u^{(m)} \in \{0, 1\}^n$ where

$$(u^{(j)})_i = \begin{cases} 1 & \text{if } j \in S_i \\ 0 & \text{otherwise} \end{cases}$$

Let $C \subseteq [n]$ be the indices of an optimal set cover for S and consider the target disjunction $f: \{0,1\}^n \to \{0,1\},\$

$$f = \bigvee_{i \in C} x_i.$$

Let \mathcal{D} be the uniform distribution over $\{u^{(1)}, \ldots, u^{(m)}, 0^n\}$. Note that given any disjunction hypothesis h for f that achieves error < 1/(m+1) under \mathcal{D} , the variables in h must constitute a set cover for \mathcal{S} .

To see why this reduction, and reductions like it, do not extend to the setting of queries, we first observe that this specific target function can be easily learned with queries, simply by querying f on all strings of Hamming weight 1. More generally and crucially, we note that the target function is defined by the optimal solution to the SETCOVER instance. While this is a very natural strategy (and indeed many other hardness results for learning employ such a strategy), for any such reduction it seems challenging to provide query access to the target function without having to solve the SETCOVER instance, which would of course render the reduction inefficient. (Beyond the issue of queries, this reduction is also limited to the inverse-polynomial error regime and does not rule out learners that are allowed larger error.) While this reduction is for the hardness of properly learning disjunctions, all aforementioned hardness results for decision trees use it as their starting point and suffer from the same limitations.

How our approach differs. Departing from these works, we design a reduction where the *definition* of our target function does not depend on the solution to a computationally hard problem—which allows us to efficiently provide the learner query access to it—and only its *decision tree complexity* scales with the quality of the solution; see Remark 2. Our resulting reduction is quite a bit more elaborate than those for the setting of random examples.

2.2 Overview of our proof and techniques

We prove Theorem 1 by reducing from VERTEXCOVER: we design an efficient mapping from graphs G to functions f where the decision tree complexity of f reflects the vertex cover complexity of G. The properties of this mapping that we require our application to learning are somewhat subtle to state, so we describe and motivate them incrementally.

2.2.1 The core reduction

Our starting point is a reduction with the following basic properties:

The core reduction

- Yes case: If G has a small vertex cover, then f has small decision tree complexity.
- \circ No case: If G requires a large vertex cover, then f has large decision tree complexity.

We sketch the main ideas behind this core reduction. For an *n*-vertex graph G, we consider its edge indicator function IsEdge_G : $\{0,1\}^n \to \{0,1\}$. An input $v = (v_1, \ldots, v_n) \in \{0,1\}^n$ to IsEdge_G is viewed as specifying the presence or absence of each vertex $v_1, \ldots, v_n \in V$, and IsEdge_G(v) = 1 iff v specifies the presence of exactly the two endpoints of some edge of G. More formally:

Definition 1 (IsEdge_G). Let G = (V, E) be an *n*-vertex graph. For an edge $e = \{v_i, v_j\} \in E$, we write $\text{Ind}[e] \in \{0, 1\}^n$ to denote its indicator string:

$$\operatorname{Ind}[e]_k = \begin{cases} 1 & \text{if } k \in \{i, j\} \\ 0 & \text{otherwise.} \end{cases}$$

The edge indicator function of G is the function IsEdge_G: $\{0,1\}^n \to \{0,1\}$,

$$\text{IsEdge}_G(v_1, \dots, v_n) = \begin{cases} 1 & (v_1, \dots, v_n) = \text{Ind}[e] \text{ for some } e \in E \\ 0 & otherwise. \end{cases}$$

When G is clear from context, we drop the subscript and simply write ISEDGE.

Warmup. We first prove:

Claim 2.1 (Decision tree complexity of IsEdge). Let G be an n-vertex m-edge graph.

• Yes case: If G has a vertex cover of size $\leq k$, then there is a decision tree T for $IsEdge_G$ of size

$$|T| \le k + m + mn.$$

• No case: If G requires a vertex cover of size $\geq k'$, then any decision tree T for IsEdge_G must have size

$$|T| \ge k' + m.$$

As stated, Claim 2.1 is not useful since the upper bound of the Yes case is much larger than the lower bound of the No case, owing to the additional additive factor of mn. More precisely, we need these bounds to satisfy:

If
$$k' = (1 + \delta)k$$
 then (Upper bound of Yes case) < (Lower bound of No case) (*)

in order to invoke the NP-hardness of $(1 + \delta)$ -approximating VERTEXCOVER.

Amplification. We therefore consider an "amplified" version of $IsEdge_G$,

 ℓ -IsEdge_G : $\{0,1\}^n \times (\{0,1\}^n)^\ell \to \{0,1\},\$

and prove:

- **Theorem 2** (Decision tree complexity of ℓ -IsEdge). Let G be an n-vertex m-edge graph and $\ell \in \mathbb{N}$.
 - Yes case: If G has a vertex cover of size $\leq k$, then there is a decision tree T for ℓ -IsEdge_G of size

$$|T| \le (\ell+1) \cdot (k+m) + mn.$$

• No case: If G requires a vertex cover of size $\geq k'$, then any decision tree T for ℓ -IsEdge_G must have size

$$|T| \ge (\ell + 1) \cdot (k' + m).$$

We point out two properties of Theorem 2 that will be important for us:

Remark 1 (Asymmetric amplification in the Yes case). Comparing Claim 2.1 and Theorem 2, we see that in No case, the entire lower bound of k' + m is amplified by a factor of $\ell + 1$. On the other hand, in the Yes case only the k + m factor—and crucially, not the mn factor—is amplified by a factor of $\ell + 1$. This is important as it allows us to choose ℓ to be sufficiently large to make the mn factor negligible, thereby having our bounds satisfy the sought-for property (*).

Remark 2 (Efficiently providing query access to ℓ -IsEdge_G). We defer the definition of ℓ -IsEdge_G to Section 5.2 but mention here that (i) it will be the hard target function in our proof of Theorem 1; and (ii) just like the unamplified IsEdge_G function—and unlike the SETCOVER-based target function described in Section 2.1— its definition will depend only on the edges in G and not its optimal vertex cover. This is crucial as it allows us to efficiently provide the learner query access to its values in our reduction without having to solve VERTEXCOVER. Circling back to our discussion in Section 2.1, this is a key qualitative difference between our reduction and previous reductions for the setting of random examples.

2.2.2 Hardness distillation

Theorem 2 already allows us to recover, with a markedly simpler proof, the best known hardness of approximation result [ZB00, Sie08] for decision tree minimization. However, it does not yet have any implications for learning since the No case only states that any decision tree that computes f exactly must have large size, and does not rule out the possibility that f can be well-approximated by a small decision tree.

We therefore strengthen the No case via a process that we call hardness distillation: we identify a small set of inputs $D \subseteq \{0,1\}^n$, which we call a coreset, that is responsible for f's large decision tree complexity.

The core reduction with hardness distillation

- Yes case: If G has a small vertex cover, then f has small decision tree complexity.
- No case: If G requires a large vertex cover, then there is a small set $D \subseteq \{0, 1\}^n$ such that any decision tree that agrees with f on D must be large.

Such a reduction yields the NP-hardness of learning decision trees to error $\langle 1/|D|$, which motivates the problem of constructing coresets that are as small as possible. Our coreset will have size poly(n), and therefore we get the hardness of learning to inverse-polynomial error. (In the next subsection we describe a further extension of this technique that establishes constant-error hardness.)

Hardness distillation via certificate complexity and relevant variables. We give a general method for identifying a small coreset that witnesses the large decision tree complexity of a function f. At a high level, there are two main components to this coreset:

- 1. A set of inputs D_1 that ensures that any decision tree T that agrees with f on D_1 must have a long path π , one of length at least s_1 .
- 2. Another set of inputs D_2 that ensures that the at-least- s_1 many disjoint subtrees that branch off of π must have sizes that sum up to at least s_2 .

See Figure 6 for an illustration. Together, D_1 and D_2 form a coreset witnessing the fact that f has decision tree complexity at least $s_1 + s_2$. To formalize this approach we rely on generalizations of two notions of boolean function complexity, namely certificate complexity and the relevance of variables, from the setting of total functions to partial functions. More formally, the two components of our method are as follows:

- 1. If there is an input $x^* \in D_1$ such that the certificate complexity of f on x^* relative to D_1 is at least s_1 , then any decision tree T that agrees with f on D_1 must have a long path π of length at least s_1 .
- 2. This path π induces at least s_1 many subfunctions of f, corresponding to f restricted by paths that diverge from π at each of π 's at-least- s_1 many nodes. If the number of variables of these subfunctions that are relevant relative to D_2 is at least s_2 , then the at-least- s_1 many disjoint subtrees that branch off π must have sizes that sum up to at least s_2 .

2.2.3 Hardness for constant error

To obtain hardness even against algorithms that are allowed constant error, we further improve the No case as follows:

The reduction for constant-error hardness

- Yes case: If G has a small vertex cover, then f has small decision tree complexity.
- No case: If G requires a large vertex cover, then there is a set $D \subseteq \{0, 1\}^n$, a distribution \mathcal{D} over D, and a constant $\varepsilon > 0$ such that any decision tree that agrees with f with probability $\geq 1 \varepsilon$ over $\boldsymbol{x} \sim \mathcal{D}$ must be large.

The key new ingredient in this final reduction is the hardness of α -Partial VERTEXCOVER, a relaxed version of VERTEXCOVER where the goal is to find a set of vertices that cover a $1-\alpha$ fraction of vertices. We show that α -PARTIALVERTEXCOVER inherits its hardness of approximation from VERTEXCOVER itself:

Claim 2.2 (Hardness of α -PARTIALVERTEXCOVER). There are constants $\alpha \in (0,1)$ and $\delta > 0$ such that if α -PARTIALVERTEXCOVER on constant-degree, n-vertex graphs can be approximated to within a factor of $1 + \delta$ in time t(n), then SAT can be solved in time $t(n \cdot \text{polylog}(n))$.

This is thanks to the fact that VERTEXCOVER is hard the approximate even for constant-degree graphs, which in turn follows from the PCP Theorem.

With Claim 2.2 in hand, Theorem 1 then follows by appropriately robustifying the other machinery described in this section.

3 Discussion and future work

Assuming SAT requires exponential time, Theorem 1 shows that the inherent time complexity of properly learning decision trees with queries is also exponential: the simple dynamic-programming-based Occam algorithm is essentially optimal, despite evidence to the contrary in the form of fast algorithms for various relaxations of the problem.

A concrete problem left open by our work is that of optimizing the efficiency of our reduction, which takes an instance of SAT over n variables and produces an instance of properly learning decision trees over $\tilde{O}(n^2)$ variables. Can this be improved to linear or quasilinear in n?

More broadly, a natural next step is to understand the complexity of *weakly-proper* learning. As mentioned in the introduction, the landscape changes dramatically for this easier setting, and we have known since the 1980s of an algorithm that runs in quasipolynomial time [EH89]. This algorithm of Ehrenfeucht and Haussler has resisted improvement for over three decades and it is reasonable to conjecture that it is in fact optimal, even for query learners:

Conjecture 1. There is no algorithm that, given queries to a size-s decision tree target and access to random labeled examples, runs in time $n^{o(\log s)}$ and returns an accurate decision tree hypothesis—one of any size, not necessarily s.

Table 1 places Theorem 1 and Conjecture 1 within the context of prior work:

	Random Examples	Queries
Proper Learning	[Ang, PV88]: Exponential lower bound. Assumption: SAT requires exponential time	Theorem 1: Exponential lower bound. Assumption: SAT requires exponential time
Weakly-proper Learning	[ABF ⁺ 09, KST23]: Quasipoly lower bound. Assumption: Inapproximability of parameterized SETCOVER	Conjecture 1: Quasipolynomial lower bound.

Table 1: Lower bounds for proper and weakly-proper learning of decision trees. In terms of upper bounds, the fastest known proper algorithm (dynamic-programming-based Occam algorithm) runs in exponential time, and the fastest known weakly-proper (Ehreunfeucht-Haussler) runs in quasipolynomial time.

Weakly-proper learning algorithms are akin to approximation algorithms, and the hardness of weakly-proper learning is akin to the hardness of approximation. An immediate, but not necessarily insurmountable obstacle in extending our techniques to the setting of weakly-proper learning is the fact that VERTEXCOVER, whose hardness of approximation we rely on in our proof, is not that hard to approximate: a simple greedy algorithm achieves a 2-approximation.

There is also more to be understood for (strongly-)proper learning of decision trees. Our work taken together with the recent query learner of [BLQT22] highlights, quite dramatically, the effect of distributional assumptions on the problem: our work gives an exponential lower bound in the distribution-free setting, whereas [BLQT22] gives an almost-polynomial time query algorithm for the uniform distribution. In the spirit of beyond worst-case analysis, an ambitious direction for future work is to understand the tractability of the problem vis-à-vis the complexity of the underlying distribution. An ultimate goal is to design efficient algorithms that circumvent the lower bounds established in this work, but nonetheless enjoy performance guarantees for the broadest possible class of distributions.

Finally, we believe that the notions of hardness distillation and coresets introduced in this work merit further study and could lead to more connections between the hardness of minimization problems and the hardness of learning.

4 Preliminaries

Notation and naming conventions. We write [n] to denote the set $\{1, 2, ..., n\}$. We use lower case letters to denote bitstrings e.g. $x, y \in \{0, 1\}^n$ and subscripts to denote bit indices: x_i for $i \in [n]$ is the *i*th index of x. The string $x^{\oplus i}$ is x with its *i*th bit flipped. We use superscripts to denote multiple bitstrings of the same dimension, e.g. $x^{(1)}, x^{(2)}, ..., x^{(j)} \in \{0, 1\}^n$. For a finite set S, Perm(S) denotes the set of permutations of S. If $S = \{s_1, ..., s_{|S|}\}$, we identify $\pi \in \text{Perm}(S)$ with the tuple $(s_{i_1}, \ldots, s_{i_{|S|}})$ where $\pi(s_j) = s_{i_j}$. In this setting, we simply write $\pi(j)$ to denote the *j*th element of the tuple, $\pi(j) = s_{i_j}$.

Distributions. We use boldface letters e.g. x, y to denote random variables. For a distribution

 \mathcal{D} , we write $\operatorname{dist}_{\mathcal{D}}(f,g) = \operatorname{Pr}_{\boldsymbol{x}\sim\mathcal{D}}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})]$. A function f is ε -close to g if $\operatorname{dist}_{\mathcal{D}}(f,g) \leq \varepsilon$. Similarly, f is ε -far from g if $\operatorname{dist}_{\mathcal{D}}(f,g) > \varepsilon$. The support of the distribution is the set of elements with nonzero mass and is denoted $\operatorname{supp}(\mathcal{D})$.

Decision trees. The size of a decision tree T is its number of internal nodes and is denoted |T|. Two subtrees of T are *disjoint* if they do not share any internal nodes. In an abuse of notation, we also write T for the function computed by the decision tree T. We say T computes a function $f : \{0,1\}^n \to \{0,1\}$ if T(x) = f(x) for all $x \in \{0,1\}^n$. The decision tree complexity of a function f is the size of the smallest decision tree computing f and is denoted DT(T).

Restrictions and decision tree paths. A restriction ρ is a set $\rho \subseteq \{x_1, \overline{x}_1, \dots, x_n, \overline{x}_n\}$ of literals, and f_{ρ} is the subfunction obtained by restricting f according to ρ : $f_{\rho}(x^*) = f(x^*|_{\rho})$ where $x^*|_{\rho}$ is the string obtained from x^* by setting its *i*th coordinate to 1 if $x_i \in \rho$, 0 if $\overline{x}_i \in \rho$, and otherwise setting it to x_i^* . We say an input x^* is consistent with ρ if $x_i \in \rho$ implies $x_i^* = 1$ and $\overline{x}_i \in \rho$ implies $x_i^* = 0$.

We identify a depth-d, non-terminal path π in a decision tree with a tuple of d literals: $\pi = (\ell_1, \ell_2, \ldots, \ell_d)$ where each ℓ_i corresponds to a query of an input variable and is unnegated if π follows the right branch and negated if π follows the left branch. Paths naturally correspond to restrictions by forgetting their ordering. Therefore, we also write f_{π} to denote the restriction of f by $\{\ell_1, \ell_2, \ldots, \ell_d\}$.

Graphs. An undirected graph G = (V, E) has n vertices $V \subseteq [n]$ and m = |E| edges $E \subseteq V \times V$. The degree of a vertex $v \in V$ is the number of edges containing it: $|\{e \in E : v \in e\}|$. The graph G is degree-d if every vertex $v \in V$ has degree at most d. We often use letters v, u, w to denote vertices of a graph G.

Learning. In the PAC learning model, there is an unknown distribution \mathcal{D} and some unknown target function $f \in \mathcal{C}$ from a fixed concept class \mathcal{C} of functions over a fixed domain. An algorithm for learning \mathcal{C} over \mathcal{D} takes as input an error parameter $\varepsilon \in (0, 1)$ and has oracle access to an example oracle $\text{EX}(f, \mathcal{D})$. The algorithm can query the example oracle to receive a pair $(\boldsymbol{x}, f(\boldsymbol{x}))$ where $\boldsymbol{x} \sim \mathcal{D}$ is drawn independently at random. The goal is to output a hypothesis h such that $\text{dist}_{\mathcal{D}}(f, h) \leq \varepsilon$. Since the example oracle is inherently randomized, any learning algorithm is necessarily randomized. So we require the learner to succeed with some fixed probability e.g. 2/3. A learning algorithm is proper if it always outputs a hypothesis $h \in \mathcal{C}$. A learning algorithm with queries is given oracle access to the target function f along with the example oracle $\text{EX}(f, \mathcal{D})$.

In this work we focus on the task of properly learning the concept class $\mathcal{T}_s = \{T : \{0,1\}^n \to \{0,1\} \mid T \text{ is a size-} s \text{ decision tree}\}.$

Definition 2 (Properly PAC learning decision trees with queries). An algorithm \mathcal{L} properly learns \mathcal{T}_s in time $t(n, s, \varepsilon)$ if for all distributions \mathcal{D} and for all $T \in \mathcal{T}_s$ and $\varepsilon \in (0, 1)$, \mathcal{L} with oracle access to $\mathrm{EX}(T, \mathcal{D})$ and queries to T runs in time $t(n, s, \varepsilon)$ and, with probability 2/3, outputs $h \in \mathcal{T}_s$ such that $\mathrm{dist}_{\mathcal{D}}(T, h) \leq \varepsilon$.

PCPs and MAX-3SAT. For this work, we are interested in reductions from SAT. Our techniques will rely on the hardness of approximation and we therefore need a reduction from SAT to approximating MAX-3SAT. The most efficient reduction exploits quasilinear PCPs:

Theorem 3 (Hardness of approximating MAX-3SAT via quasilinear PCPs [Din07, BSS08]). There is a constant $c \in (0,1)$ and a polynomial-time reduction that takes a 3CNF formula φ with m clauses and produces a 3CNF formula φ^* with $O(m \cdot \text{polylog}(m))$ clauses satisfying

- if φ is satisfiable then φ^* is satisfiable;
- if φ is unsatisfiable then no assignment satisfies a c-fraction of clauses of φ^* .

4.1 Hardness of Vertex Cover

Vertex cover. A vertex cover for an undirected graph G = (V, E) is a subset of the vertices $C \subseteq V$ such that every edge has at least one endpoint in C. We write $VC(G) \in \mathbb{N}$ to denote the size of the smallest vertex cover. See Figure 1 for an example of a vertex cover. The VERTEXCOVER problem is to decide whether a graph contains a vertex cover of size-k, i.e. to decide if $VC(G) \leq k$. We consider the more general gapped vertex problem where the problem is to decide whether a graph has a small vertex cover or requires large vertex cover. Specifically we write (k, k')-VERTEXCOVER for the problem of deciding whether a graph contains a vertex cover of size-k or every vertex cover has size at least k'. This gapped problem is equivalent to the problem of approximating vertex cover. There is a polynomial-time greedy algorithm for vertex cover that approximates it within a factor of 2, i.e. solves (k, 2k)-VERTEXCOVER in polynomial-time.

Constant factor hardness of VERTEXCOVER is known, even for bounded degree graphs (graphs whose degree is bounded by some universal constant). Papadimitriou and Yannakakis in [PY91] give an approximation preserving reduction from MAX-3SAT to VERTEXCOVER on constant-degree graphs. The PCP theorem [AS98, ALM⁺98] implies NP-hardness of approximating MAX-3SAT and therefore, combined with the reduction in [PY91], implies hardness of approximating VERTEX-COVER on constant-degree graphs. (For a further discussion and history of these results, see the survey by Trevisan [Tre14].)

Theorem 4 (Hardness of approximating VERTEXCOVER). There are constants $\delta > 0$ and $d \in \mathbb{N}$ such that if $(k, (1 + \delta) \cdot k)$ -VERTEXCOVER on *n*-vertex degree-*d* graphs can be solved in time t(n), then SAT can be solved in time $t(n \cdot \operatorname{polylog}(n))$.

This hardness follows from Theorem 3 and the reduction in [PY91]. The $n \cdot \text{polylog}(n)$ factor originates from Theorem 3.

The fact that Theorem 4 holds for constant degree graphs will be essential for our lower bound because it allows us to assume that k is large: $VC(G) = \Theta(m)$.

Fact 4.1 (Constant degree graphs require large vertex covers). If G is an m-edge degree-d graph, then $VC(G) \ge m/d$.

This fact follows from the observation that in a degree-d graph each vertex can cover at most d edges.

5 A reduction from VertexCover to Decision Tree Minimization

5.1 Intuition and warmup: the $IsEdge_G$ function

In this section we prove Claim 2.1, which serves as a warmup for our core reduction, Theorem 2. We first introduce a few notions (and notation) that will be useful throughout the rest of the paper.



Figure 1: A graph G = (V, E) with 10 edges having VC(G) = 3. The unique vertex cover of size 3 is highlighted in teal.

5.1.1 Useful notions and notation: edge partitions and divergent path prefixes

Edge partitions induced by decision trees for IsEdge. We will make use of the notion of a *restricted edge neighborhood* and a *restricted vertex neighborhood*. Specifically, we will be interested in the edges incident to a particular vertex which do *not* contain certain vertices.

Definition 3 (Restricted edge and vertex neighborhood). For a graph G = (V, E), the edge neighborhood of $v_{i_{\kappa}} \in V$ restricted by $v_{i_1}, \ldots, v_{i_{\kappa-1}}$, denoted $E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$, is the set of edges containing $v_{i_{\kappa}}$ but not any of $v_{i_1}, \ldots, v_{i_{\kappa-1}}$:

$$E(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}}) \coloneqq \{e \in E \mid v_{i_{\kappa}} \in e \text{ and } v_{i_1}, \dots, v_{i_{\kappa-1}} \notin e\}.$$

The vertex neighborhood of $v_{i_{\kappa}}$ restricted by $v_{i_1}, \ldots, v_{i_{\kappa-1}}$, denoted $V(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$, is the set of neighbors of $v_{i_{\kappa}}$ excluding the vertices $v_{i_1}, \ldots, v_{i_{\kappa-1}}$:

$$V(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}}) \coloneqq \left\{ v \in V \mid \{v_{i_{\kappa}}, v\} \in E \text{ and } v \neq v_{i_1}, \dots, v_{i_{\kappa-1}} \right\}.$$

Often when a tuple of vertices $(v_{i_1}, \ldots, v_{i_k})$ is understood from context, we will use the shorthand notation $E_{\kappa} = E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ for $\kappa = 1, \ldots, k$ and likewise for V_{κ} . Restricted edge and vertex neighborhoods are closely related to each other, and each can be defined in terms of the other:

$$E_{\kappa} = \{\{v_{i_{\kappa}}, v\} \mid v \in V_{\kappa}\} \quad \text{and} \quad V_{\kappa} = \{v \mid \{v_{i_{\kappa}}, v\} \in E_{\kappa}\}.$$

Given a vertex cover $\{v_{i_1}, \ldots, v_{i_k}\}$, the sets $\{E_\kappa\}_{\kappa \in [k]}$ form a partition of the edge set E. Indeed,

$$\bigcup_{\kappa \in [k]} E_{\kappa} = E$$

since every edge in G is incident to some vertex $v_{i_{\kappa}}$. Also, the sets E_{κ} are disjoint since each E_{κ} excludes the edges already covered by the previous $E_1, \ldots, E_{\kappa-1}$ sets. In fact, the converse also holds. If $C = \{v_{i_1}, \ldots, v_{i_k}\}$ are vertices such that E_{κ} partition the edge set then C must form a vertex cover: every edge $e \in E$ is in some partition E_{κ} and so $v_{i_{\kappa}}$ covers e.

Fact 5.1. Let $C = \{v_{i_1}, \ldots, v_{i_k}\}$ be a subset of vertices of a graph G and $E_{\kappa} \coloneqq E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ for $\kappa \in [k]$. Then C forms a vertex cover of G if and only if $\{E_{\kappa}\}_{\kappa \in [k]}$ form a partition of E.

A key property of the IsEdge_G function is that every decision tree for it induces such an edge partition in the following way. Every decision tree for IsEdge_G has a path π in it whose path variables form a vertex cover. This vertex cover induces a partition of the edges of G. Each part of the partition corresponds to a unique variable in this decision tree path. This correspondence will be important for lower bounding the size of the decision tree in the case when G requires large vertex covers. To describe this correspondence, it will be useful for us to have the following notation for a path that diverges from from π at a particular point and then stops. **Definition 4** (Divergent path prefix; see Figure 2). For a path π , the path $\pi|_{\oplus\kappa}$ denotes the path which follows π for the first $\kappa - 1$ queries, flips the κ th query, then terminates:



Figure 2: Illustration of a divergent path prefix. The root-to-leaf path π is illustrated in purple. At depth κ the path $\pi|_{\kappa}$ diverges and terminates.

If π is the path corresponding to a vertex cover, then $\pi|_{\oplus\kappa}$ corresponds to the path followed by edges in E_{κ} (here we are conflating edges and edge indicator strings).

5.1.2 **Proof of Claim 2.1**

Proof of the Yes case. Let $C = \{v_{i_1}, \ldots, v_{i_k}\}$ be a vertex cover for G. The leftmost branch π of our decision tree queries these vertices successively and terminates with a 0-leaf. These are the vertices colored blue in Figure 3.

We move on to describing each of the subtrees branching off of π . More formally, for each $\kappa \in [k]$, we describe the subtree rooted at the end of $\pi|_{\oplus\kappa}$ (i.e. the subtree that is the 1-successor of v_{i_k}). At this point T "knows" that v_{i_k} is set to 1. For IsEdge to output 1, exactly one of v_{i_κ} 's neighbors must also be set to 1, and all n-2 other vertices must be set to 0 (i.e. these are precisely the inputs $\operatorname{Ind}[e]$ for $e \in E_{\kappa}$). Therefore T queries all $v \in V_{\kappa}$ (i.e. the neighbors of $v_{i_{\kappa}}$ that have not already been queried along π), testing to see whether any of them are 1, and terminates with a 0-leaf if they are all set to 0. These are the vertices colored teal in Figure 3.

Finally, we describe the subtree that is the 1-successor of each $v \in V_{\kappa}$. At this point T knows that $v_{i_{\kappa}}$ and this neighbor v are both set to 1, and it remains only to check that all other vertices are set to 0 before outputting 1: it queries all n-2 vertices in V that are not v or $v_{i_{\kappa}}$ and outputs 1 iff all of them are set to 0. These are the vertices colored orange in Figure 3.

We complete the proof by bounding the size of T. Its leftmost branch has size k (the blue vertices). By Fact 5.1, querying all $v \in V_{\kappa}$ for $\kappa \in [k]$ results in an additional $\sum_{\kappa} |V_{\kappa}| = \sum_{\kappa} |E_{\kappa}| = m$ internal nodes (the teal vertices). After each of these m internal nodes, we query n-2 more vertices, resulting in an additional m(n-2) < mn internal nodes (the orange vertices). Thus, the total size of T is at most k + m + mn.



Figure 3: An illustration of the proof of the Yes case of Claim 2.1. Given a vertex cover $C = \{v_{i_1}, \ldots, v_{i_k}\}$ of G, our decision tree for IsEdge queries C among the leftmost branch (colored blue in the figure). If some vertex $v_{i_{\kappa}} \in C$ is set to 1 then the decision tree queries all vertices in V_{κ} (colored teal). Once some $v \in V_{\kappa}$ is set to 1, the decision tree queries the remaining unqueried vertices to ensure that they are set to 0 (colored orange) before outputting 1.

We proceed to a proof of the lower bound.

Proof of the No case. Our proof consists of two parts: (1) proving that the leftmost branch of T must be a vertex cover and therefore has size at least k' and (2) showing that the rest of the tree has size at least m. See Figure 4 for an illustration.

- 1. Leftmost branch must be a vertex cover. Let π be the leftmost branch of T and suppose for contradiction that the vertices queried along π do not form a vertex cover for G. This means that there is some edge $e \in E$ that is not queried along π , and hence both $\operatorname{Ind}[e]$ and 0^n will follow π and reach the same leaf. Since $\operatorname{IsEdge}(0^n) = 0 \neq 1 = \operatorname{IsEdge}(\operatorname{Ind}[e])$, this is a contradiction.
- 2. Rest of the tree has at least m nodes. Let us order the vertices of π from root downwards as $v_{i_1}, \ldots, v_{i_{|\pi|}}$. For each $\kappa \in [|\pi|]$, we consider the subtree T_{κ} that is the 1-successor of v_{κ} . Consider $e \in E_{\kappa}$ and suppose $e = (v_{i_{\kappa}}, v)$. By the definition of E_{κ} , the endpoint vhas not yet been queried when $\operatorname{Ind}[e]$ enters T_{κ} . Thus, T_{κ} must query v, since otherwise

T cannot distinguish between $\operatorname{Ind}[e]$ and $\operatorname{Ind}[e]^{\oplus v}$ (note that $\operatorname{IsEdge}(\operatorname{Ind}[e]) = 1 \neq 0 = \operatorname{IsEdge}(\operatorname{Ind}[e]^{\oplus v})$). Further, all $e \in E_{\kappa}$ will have distinct second endpoints that T_{κ} must query (since if not, then they would share both their endpoints and be the exact same edge). In other words, we have argued that T_{κ} must query all the vertices in V_{κ} .

Since the sets E_{κ} for $\kappa \in [|\pi|]$ partition the edges (Fact 5.1), we have that all these disjoint subtrees $T_1, \ldots, T_{|\pi|}$ taken together must query at least $\sum_{\kappa} |V_{\kappa}| = |E_{\kappa}| = |E| = m$ additional vertices.

Combining the two claims above we show shown that $|T| \ge k' + m$ and the proof is complete. \Box



Figure 4: An illustration of the No case of Claim 2.1. Given any decision tree T computing IsEdge, the leftmost branch π must form a vertex cover of G. Furthermore, for each vertex $v_{i_{\kappa}}$ queried along π , the subtree T_{κ} branching off of π at $v_{i_{\kappa}}$ must query all the vertices in V_{κ} . The size of T is therefore at least $k' + \sum_{\kappa} |V_{\kappa}| = k' + m$.

5.2 *l*-IsEdge: an amplified version of ISEDGE

Definition 5 (The ℓ -amplified ISEDGE function). Let G = (V, E) be an *n*-vertex graph and $\ell \in \mathbb{N}$. The ℓ -amplified edge indicator function of G is the function

$$\ell$$
-IsEdge_G : $\{0,1\}^n \times (\{0,1\}^n)^\ell \to \{0,1\}$

defined as follows: ℓ -IsEdge_G $(v^{(0)}, v^{(1)}, \dots, v^{(\ell)}) = 1$ iff

- 1. IsEdge_G $(v^{(0)}) = 1$ (i.e. $v^{(0)} = \text{Ind}[e]$ for some $e \in E$), and
- 2. $v_i^{(1)} = \dots = v_i^{(\ell)} = 1$ for all $i \in [n]$ such that $v_i^{(0)} = 1$.

Notation and terminology. When G is clear from context, we drop the subscript and just write ℓ -ISEDGE. We also use $N := n + n\ell$ to denote the number of inputs to ℓ -ISEDGE. We refer to $v_1^{(0)}, \ldots, v_n^{(0)}$ as the original variables. As in the nonamplified IsEdge function, there is a natural correspondence between these original variables and the vertices $V = \{v_1, \ldots, v_n\}$ of G. For each original variable $v_i^{(0)}$, we refer to $v_i^{(1)}, \ldots, v_i^{(\ell)}$ as its duplicated variables and write

$$\mathrm{Dup}(v_i) \coloneqq \left\{ v_i^{(1)}, \dots, v_i^{(\ell)} \right\}.$$

We write ℓ -Ind $[e] \in (\{0,1\}^n)^{\ell+1}$ to denote the string (Ind $[e], \ldots,$ Ind[e]). Note that ℓ -IsEdge(ℓ -Ind[e]) = 1 and these are the 1-inputs of minimum Hamming weight.

Asymmetries in the definition of ℓ -IsEdge. We note two sources of asymmetry in the definition of ℓ -IsEdge, both of which are crucial for Theorem 2 (specifically, Remark 1) to hold. First, the original variables play a distinct role from the duplicated ones: for ℓ -IsEdge to output 1, the original variables have to agree with an edge indicator but the duplicated variables do not. Second, there is also an asymmetry between 1- and 0-coordinates: for ℓ -IsEdge to output 1, the duplicated variables have to be set to 1 whenever the original variables are set to 1, but the same is not true for the 0-coordinates.

5.2.1 Proof of Theorem 2



Figure 5: An illustration of the proof of the Yes case of Theorem 2. Given a vertex cover $C = \{v_{i_1}, \ldots, v_{i_k}\}$ of G, our decision tree for ℓ -IsEdge queries the original variables corresponding to C among the leftmost branch (colored blue in the figure). If some vertex $v_{i_{\kappa}}^{(0)} \in C$ is set to 1 then the decision tree queries all vertices in $\text{Dup}(v_{i_{\kappa}}^{(0)})$ (colored in teal). If all of these are 1, then it proceeds to compute the appropriate IsEdge subfunction on the remaining vertices.

Proof of the Yes case. The construction is a slight extension of our tree for IsEdge that we constructed for the Yes case of Claim 2.1. See Figure 5 for an illustration of this construction. Let $C = \{v_{i_1}, \ldots, v_{i_k}\}$ be a vertex cover of G. Similar to before, the leftmost branch π of our tree Tqueries the original variables $v_{i_1}^{(0)}, \ldots, v_{i_k}^{(0)}$ corresponding to these vertices and terminates with a 0-leaf. We now describe the subtree T_{κ} that is the 1-successor of $v_{i_{\kappa}}^{(0)}$ for $\kappa \in [k]$. It first checks if the duplicated variables of $v_{i_{\kappa}}$ are all set to 1, since that is a necessary criterion for ℓ -IsEdge to output 1: it queries all ℓ variables in $\text{DUP}(v_{i_{\kappa}})$ and outputs 0 once any of them are set to 0. If all of them are indeed set to 1, then for T_{κ} to output 1 it must check that (i) there is a neighbor v of $v_{i_{k}}$ whose original variable is set to 1, (ii) all the other original variables are set to 0, and (iii) the duplicated variables of v are set to 1.

For (i) and (ii), T_{κ} queries the remaining original variables in a manner identical to the tree for IsEdge. Briefly restating that construction, it verifies Condition (i) by querying the original variables of $v \in V_{\kappa}$, testing to see of any of them are set to 1. If none of them are set to 1, it outputs 0. Otherwise, once the original variable of some $v \in V_{\kappa}$ is set to 1, the tree T_{κ} moves on to verifying Condition (ii): it queries the original variables of all n - 2 vertices in $V \setminus \{v_{i_{\kappa}}, v\}$ and outputs 0 once any of them are set to 1. If all of them are indeed set to 0, it moves on to verifying Condition (iii). It queries all ℓ variables in DUP(v) and outputs 1 iff all of them are set to 1.

We now bound the size of this tree. The portion of it that is identical to the tree for IsEdge will have size at most k + m + mn, as proved in Claim 2.1. We incur an additional $k\ell$ nodes to query the duplicate variables for the vertex cover: ℓ duplicate variables for each vertex in the size-kvertex cover. Finally, since we additionally query the ℓ many duplicate variables for each $v \in V_{\kappa}$ in the subtree T_{κ} , we incur another additional $\ell \sum_{\kappa} |V_{\kappa}| = \ell \sum_{\kappa} |E_{\kappa}| = \ell m$ many nodes. In total, this results in a tree of size

$$|T| \le k + m + mn + k\ell + \ell m = (\ell + 1)(k + m) + mn.$$

We now prove the lower bound.

Proof of No case. Just as in the proof of Claim 2.1, we divide our proof into two parts. We show that (1) the leftmost branch of any decision tree for ℓ -ISEDGE must correspond to a vertex cover and hence has size at least k', and (2) the rest of the tree must have size at least $\ell k' + (\ell + 1)m$.

- 1. Leftmost branch must be a vertex cover. Let π be the leftmost branch of T and $v_{i_1}^{(j_1)}, \ldots, v_{i_{|\pi|}}^{(j_{|\pi|})}$ be the variables queried along π . We claim that the corresponding vertices $v_{i_1}, \ldots, v_{i_{|\pi|}} \in V$ must form a vertex cover for G. Suppose for contradiction that they do not. This means that there is some edge $e \in E$ such that neither the original nor duplicated variables of e's endpoints are queried along π . Therefore both ℓ -Ind[e] and 0^N will follow π and reach the same leaf. Since IsEdge $(0^N) = 0 \neq 1 = \ell$ -IsEdge $(\ell$ -Ind[e]), this is a contradiction.
- 2. Rest of the tree has at least $\ell k' + (\ell+1)m$ nodes. Order the vertices of π from root downwards as $v_{i_1}, \ldots, v_{i_{|\pi|}}$. We will consider only the indices $\kappa \in [|\pi|]$ such that E_{κ} is nonempty, noting that the vertices corresponding to these indices still form a vertex cover, and hence there are at least k' such indices. Fix such a κ and consider the subtree T_{κ} that is the 1-successor of $v_{i_{\kappa}}^{(j_{\kappa})}$. Let $e = (v_{i_{\kappa}}, v) \in E_{\kappa}$. We argue that $v_{i_{\kappa}}$ is responsible for ℓ additional queries within T_{κ} , and v for $\ell + 1$ additional ones. For the former claim, let $j \in \{0, \ldots, \ell\} \setminus \{j_{\kappa}\}$. By the definition of E_{κ} , the variable $v_{i_{\kappa}}^{(j)}$ has not yet been queried when ℓ -Ind[e] enters T_{κ} . Since

$$\ell\text{-IsEdge}(\ell\text{-Ind}[e]) = 1 \neq 0 = \ell\text{-IsEdge}(\ell\text{-Ind}[e]^{\oplus v_{i_{\kappa}}^{(j)}}),$$

it follows that $v_{i_{\kappa}}^{(j)}$ must be queried within T_{κ} . Similarly, the latter claim follows from the fact that T_{κ} must query the original and all the duplicated variables of v, a total of $\ell + 1$ many

variables. This latter claim holds for all endpoints of edges $e \in E_{\kappa}$ (i.e. the vertices $v \in V_{\kappa}$), so in total T_{κ} must contain at least $\ell + (\ell + 1)|V_{\kappa}|$ nodes. Summing over all $\kappa \in [|\pi|]$ such that E_{κ} is nonempty and applying Fact 5.1, we get that the disjoint subtrees $T_1, \ldots, T_{|\pi|}$ must query at least

$$\sum_{\kappa: E_{\kappa} \neq \emptyset} \ell + (\ell+1)|E_{\kappa}| = \ell k' + (\ell+1)m$$

many variables.

Combining the two claims above we have shown that

$$T| \ge k' + \ell k' + (\ell + 1)m = (\ell + 1)(k' + m)$$

and the proof is complete.

5.3 Hardness of decision tree minimization

DT-MIN: Given a decision tree $T^* : \{0,1\}^n \to \{0,1\}$, construct a minimum-size decision tree T such that $T \equiv T^*$ (i.e. $T(x) = T^*(x)$ for all $x \in \{0,1\}^n$).

This problem of decision tree minimization was first shown to be NP-hard by Zantema and Bodlaender [ZB00]. That result was subsequently improved by Sieling [Sie08] who showed that the problem is even NP-hard to approximate. Using Theorem 2 we recover this hardness of approximation. We begin by observing that our proofs of the Yes and No cases of Theorem 2 are algorithmic in the following sense:

- In the Yes case, we showed that given a graph G and a size-k vertex cover, the tree T for ℓ -IsEdge of size $(\ell + 1) \cdot (k + m) + mn$ can be constructed in $poly(\ell, n)$ time.
- In the No case, we showed that given a size-s' tree T for ℓ -IsEdge, a size-k' vertex cover for G satisfying $(\ell + 1) \cdot (k' + m) \leq s'$ can be constructed in $poly(\ell, n)$ time.

With these observations in hand, we are now ready to recover [Sie08]'s result.

Lemma 5.2 (A reduction from VERTEXCOVER to DT-MIN). There is a polynomial-time reduction that takes a degree-d, n-vertex, m-edge graph G and produces a decision tree T^* such that the following holds. Given any tree T such that $T \equiv T^*$ and whose size is within a $(1 + \delta)$ factor of the optimal for T^* , one can construct in polynomial time a size-k' vertex cover of G satisfying $k' \leq (1 + \delta') \cdot VC(G)$ where $\delta' = O(d\delta)$.

Proof. Let $\ell \coloneqq 2mn$. We begin by applying the Yes case of Theorem 2 to G with the trivial vertex cover of all n vertices to obtain a decision tree T^* for ℓ -IsEdge of size

$$(\ell + 1) \cdot (n + m) + mn = (2mn + 1) \cdot (n + m) + mn.$$

As observed above, our proof of Theorem 2 shows that T^* can be constructed from G in poly(n) time. This tree T^* will be the input to DT-MIN in our reduction.

By the Yes case of Theorem 2 again, if VC(G) =: k then

$$DT(\ell\text{-IsEdge}) \le (\ell+1)(k+m) + mn \rightleftharpoons s.$$

Suppose an algorithm for DT-MIN returns a tree T for ℓ -IsEdge of size s' where $s' \leq (1 + \delta) \cdot s$. We claim that we can then efficiently construct a vertex cover for G of size k' where $k' \leq (1 + \delta') \cdot k$ and $\delta' = O(d\delta)$, thereby completing the reduction. Our proof of Theorem 2 shows that we can efficiently construct from T, in poly(n) time, a size-k' vertex cover satisfying:

$$(\ell+1)(k'+m) \le s'.$$

The claim that $s' \leq (1 + \delta) \cdot s$ is therefore equivalent to

$$(\ell + 1) \cdot (k' + m) \le (1 + \delta) \cdot [(\ell + 1)(k + m) + mn].$$

Rearranging the above, we get that

$$\begin{aligned} k' &\leq (1+\delta) \cdot k + \delta m + \frac{(1+\delta) \cdot mn}{\ell+1} \\ &\leq (1+\delta) \cdot k + \delta kd + \frac{(1+\delta) \cdot mn}{\ell+1} \\ &< (1+\delta) \cdot k + \delta kd + 1 \\ &< [1+\delta(d+2)] \cdot k \end{aligned}$$
 (Our choice of ℓ)

and the proof is complete.

[Sie08]'s result now follows as an immediate consequence of Lemma 5.2 and the fact that VER-TEXCOVER is hard to approximate even for constant-degree graphs (Theorem 4):

Theorem 5 (Hardness of approximation for DT-MIN [Sie08]). There is a constant $\delta \in (0, 1)$ such that if DT-MIN can be approximated to within a factor of $1 + \delta$ in polynomial-time, then P = NP.

([Sie08] then amplifies this constant-factor inapproximability to a superconstant factor using an XOR lemma from [HJLT96]. We refer the interested reader to [Sie08] for the details of this step.)

In the next section, we strengthen [Sie08]'s result by showing that the same hardness holds even if the algorithm need only minimize T over a small set of input points rather than all of $\{0, 1\}^n$.

6 Hardness distillation and learning consequence for small error

6.1 A general method for hardness distillation

For a function $f : \{0,1\}^n \to \{0,1\}$, the quantity DT(f) captures the complexity of computing fon all of its inputs. If DT(f) is large, then any small decision tree that tries to compute f must err on at least one point in $\{0,1\}^n$. For some f, it may be the case that, more specifically, there is a fixed set $D \subseteq \{0,1\}^n$ such that all small decision trees err on some point in D. The set D then captures or "distills" the hardness of f since any function g which agrees with f over the set Dmust also have large decision tree complexity. We call this set D a coreset.¹ Ultimately, our goal

¹This naming convention is inspired by, though not formally related to, the notion of a coreset from the clustering literature.

will be to identify explicitly coresets D which distill the hardness of the target function f. This way, any learner that learns f over the distribution $\operatorname{Uniform}(D)$ to $\operatorname{error} < \frac{1}{|D|}$ has to output a decision tree whose size captures $\operatorname{DT}(f)$. Since the error scales with $\frac{1}{|D|}$, we have a vested interest in making D has small as possible so that we can tolerate large learning errors. In this section, we identify a general method for distilling the hardness of a function f into a coreset D. We start by generalizing certificate complexity and relevant variables with respect to fixed subsets D.

Certificate complexity with respect to a set of inputs. A certificate for $f : \{0, 1\}^n \to \{0, 1\}$ over a set of inputs $D \subseteq \{0, 1\}^n$ on $x \in \{0, 1\}^n$ is a restriction ρ consistent with x such that $f_{\rho}(y) = f_{\rho}(x)$ for all $y \in D$. The certificate complexity of x on f over D is the size of the smallest certificate of x on f over D.

One useful fact is that a decision tree path forms a certificates for the inputs that follow it.

Fact 6.1 (Decision tree paths are certificates). If a decision tree T computes $f : \{0,1\}^n \to \{0,1\}$ over $D \subseteq \{0,1\}^n$, then the path that an input $x \in D$ follows in T forms a certificate for f over D on x.

Indeed, in the above, any root-to-leaf path π terminates in a leaf which implies f_{π} is a constant function over D. Any input $x \in D$ that follows π is consistent with it and so $f(x) = f_{\pi}(x) = f_{\pi}(y)$ for all $y \in D$.

Relevant variables with respect to a set of inputs. A variable $i \in [n]$ is said to be relevant for $f : \{0,1\}^n \to \{0,1\}$ over a set of inputs $D \subseteq \{0,1\}^n$ if there is some $x \in D$ such that $x^{\oplus i} \in D$ and $f(x) \neq f(x^{\oplus i})$. We write $\operatorname{Rel}(f; D) \in [n]$ for the number of relevant variables of f with respect to D. When referring to the number of relevant variables over the entire domain $D = \{0,1\}^n$, we drop D and simply write $\operatorname{Rel}(f)$. If $D \subseteq D'$, then every relevant variable for f over D is also relevant for f over D'. Therefore, $\operatorname{Rel}(f, D) \leq \operatorname{Rel}(f, D')$ and in particular $\operatorname{Rel}(f, D) \leq \operatorname{Rel}(f)$ for all D.

Decision tree complexity with respect to a set of inputs. The decision tree complexity of $f : \{0,1\}^n \to \{0,1\}$ over $D \subseteq \{0,1\}^n$ is the size of the smallest decision tree that computes f over D and is denoted DT(f,D). Any decision tree that computes f also computes f over D and so $DT(f,D) \leq DT(f)$.

We now state and prove the main result for this section.

Theorem 6 (Hardness distillation). Let $f : \{0,1\}^n \to \{0,1\}$ and $D \subseteq \{0,1\}^n$ be a set of inputs. Let $s_1 \in \mathbb{N}$ lower bound the certificate complexity of x on f over D. Let $s_2 \in \mathbb{N}$ satisfy

$$\sum_{i=1}^{|\rho|} \operatorname{Rel}(f_{\pi|_{\oplus i}}; D) \ge s_2$$

for every certificate ρ for x on f over D and $\pi \in \text{Perm}(\rho)$, a permutation of ρ . Then,

$$\mathrm{DT}(f,D) \ge s_1 + s_2.$$



Figure 6: An illustration of hardness distillation for a function f. A path π through the decision tree is highlighted in purple. This path corresponds to an ordering of a certificate for some input x that follows this path. The subtrees hanging off the main path π compute the subfunctions $f_{\pi|\oplus i}$ where $\pi|_{\oplus i}$ corresponds to the path leading to the root of the subtree. The sum of the number of relevant variables of these subfunctions plus the length of the path π lower bounds the overall size of the decision tree.

If one can show for some D that the quantity $s_1 + s_2$ captures the decision tree complexity of f, then D is a good candidate for hardness distillation. Figure 6 illustrates some intuition for the quantity $\sum_{i=1}^{|\rho|} \operatorname{Rel}(f_{\pi|\oplus i}; D)$ in Theorem 6. If $x \in D$, then any decision tree for f over D contains a certificate, ρ , for f on x. The depth- $|\rho|$ path followed by x induces an ordering over ρ and naturally yields $|\rho|$ disjoint subtrees, each of which hangs off the main path. The size of the main decision tree is lower bounded by the sizes of these subtrees plus the length of the path followed by x. The sizes of these subtrees can be lower bounded by the number of relevant variables of the corresponding subfunctions which then yields the desired lower bound.

Before proving Theorem 6, we establish a lemma stating that the number of relevant variables of disjoint subtrees of a decision tree lower bounds its size.

Lemma 6.2 (Relevant variables of disjoint subtrees lower bound decision tree size). Let T be a decision tree, and let T_1, \ldots, T_d be disjoint subtrees of T. Then,

$$|T| \ge \sum_{i=1}^{d} \operatorname{Rel}(T_i).$$

Proof. If a variable x_j is *not* queried in the subtree T_i , then x_j cannot be relevant for the function T_i . Indeed, in this case, the leaf in T_i that any input x reaches is the same as the leaf that $x^{\oplus j}$ reaches. Therefore, every relevant variable of T_i is queried in the subtree. Since the subtrees T_1, \ldots, T_d are disjoint, each relevant variable of T_i can be mapped to a *unique* internal node of T.

It follows that

$$|T| \ge \sum_{i=1}^{d} |T_i| \ge \sum_{i=1}^{d} \operatorname{Rel}(T_i).$$

With this lemma in hand, we are able to prove Theorem 6.

Proof of Theorem 6. Let T be any decision tree computing f over D. We will show that $|T| \geq s_1 + s_2$. Let π be the path followed by $x \in D$ in T. By Fact 6.1, π is a certificate for f over D on x. Therefore $|\pi| \geq s_1$. Recall from Definition 4 that $\pi|_{\oplus i} = \left\{\pi(1), \ldots, \pi(i-1), \overline{\pi(i)}\right\}$ corresponds to the depth i path in T that follows x to depth i-1 and then diverges from x on the ith variable queried. Let $T_{\pi|_{\oplus i}}$ denote the subfunction of T computed by the subtree rooted at the last variable queried in $\pi|_{\oplus i}$. Each $T_{\pi|_{\oplus i}}$ contributes $\operatorname{Rel}(T_{\pi|_{\oplus i}})$ many variables to the size of T by Lemma 6.2 and the path ρ itself contributes at least s_1 many variables since π is also disjoint from the subtrees. It follows that

$$|T| \ge |\pi| + \sum_{i=1}^{|\pi|} \operatorname{Rel}(T_{\pi|_{\oplus i}})$$

$$\ge s_1 + \sum_{i=1}^{|\pi|} \operatorname{Rel}(T_{\pi|_{\oplus i}}; D)$$

$$= s_1 + \sum_{i=1}^{|\pi|} \operatorname{Rel}(f_{\pi|_{\oplus i}}; D)$$

$$(T \text{ computes } f \text{ over } D)$$

(Assumption from theorem statement)

6.2 Warmup: hardness distillation for IsEdge

 $\geq s_1 + s_2.$

We start by applying the framework from Section 6.1 to the function ISEDGE. The first step is to identify a small coreset D which captures decision tree size.

Definition 6 (Decision tree coreset of the IsEdge function). For an *n*-vertex graph G, the set $D_G \subseteq \{0,1\}^n$ consists of the points

- all edge indicators: $\operatorname{Ind}[e] \in \{0,1\}^n$ such that $e \in E$;
- all 1-coordinate perturbations of edge indicators: $\operatorname{Ind}[e]^{\oplus i}$ and $\operatorname{Ind}[e]^{\oplus j}$ for all $e = \{v_i, v_j\} \in E$;
- the all 0s inputs: 0^n .

Example. See Figure 7 for an example of a graph G and the associated set of inputs D_G .



Figure 7: Example of a graph G on four vertices and the associated set of inputs $D_G \subseteq \{0,1\}^4$. Each row in the table corresponds to a data point in D_G . The first 4 rows correspond to the edges in G and are color coded to highlight which row corresponds to which edge. The next 4 rows correspond to 1-coordinate perturbations of the edge indicators, all of which are Hamming neighbors of edge indicators.

Recall from Claim 2.1 that $DT(IsEdge_G) \ge k' + m$ where k' is the size of a vertex cover for G. The main claim of this section is that D_G "distills" this hardness factor of $IsEdge_G$. The upper bound from Claim 2.1 immediately applies to D_G . That is, $k + m + mn \ge DT(IsEdge_G) \ge DT(IsEdge_G, D_G)$. Therefore, the lower bound is all that remains for showing D_G is a good coreset.

Claim 6.3 (D_G is a decision tree coreset for IsEdge). Let G be an m-vertex graph. Then

 $DT(IsEdge, D_G) \ge k' + m$

where k' is the size of a vertex cover for G.

Ultimately, we would like to prove Claim 6.3 by applying Theorem 6 where f is the function IsEdge, D is the set of inputs D_G , and x is the input 0^n . To this end, we first show that k' is a lower bound on the certificate complexity of 0^n on IsEdge over D_G . Then we prove a lemma showing that the number of edges in G lower bounds the number of relevant variables of subfunctions of IsEdge induced by certificates of 0^n .

Proposition 6.4 (Any certificate of 0^n contains a vertex cover). Let G be an n-vertex graph and let ρ be a certificate for IsEdge over D_G on 0^n . Then the variables in ρ form a vertex cover of G.

Proof. If the variables in ρ do not cover some edge $e \in E$, then $\operatorname{Ind}[e] \in \{0,1\}^n$ is consistent with ρ and $\operatorname{IsEdge}_G(0^n) = 0 \neq 1 = \operatorname{IsEdge}_G(\operatorname{Ind}[e])$ implies ρ is not a certificate. Therefore, any certificate ρ must contain a vertex cover.

Lemma 6.5 (Lower bounding the number of relevant variables of IsEdge subfunctions). Let G be an n-vertex graph, ρ a certificate for IsEdge : $\{0,1\}^n \to \{0,1\}$ over D_G , and $\pi = (\overline{v}_{i_1}, \ldots, \overline{v}_{i_k}) \in \text{Perm}(\rho)$ a permutation of ρ . Then

$$\operatorname{Rel}(\operatorname{IsEdge}_{\pi|_{\oplus\kappa}}; D_G) \ge |E(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}})|$$

for all $\kappa \in [k]$.

Proof. Let $\pi|_{\oplus\kappa}$ be as in the lemma statement and let $v \in V_{\kappa} = V(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ be arbitrary (recall the definition of these quantities from Definitions 3 and 4). Let $e = (v_{i_{\kappa}}, v) \in E_{\kappa} =$ $E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ be the edge containing v. The input $\operatorname{Ind}[e] \in D_G$ has a 1 for the coordinates corresponding to v and $v_{i_{\kappa}}$ and 0s elsewhere. Therefore, it is consistent with $\pi|_{\oplus\kappa} = \{\overline{v}_{i_1}, \ldots, \overline{v}_{i_{\kappa-1}}, v_{i_{\kappa}}\}$ (since $v \notin \{v_{i_1}, \ldots, v_{i_{\kappa-1}}\}$ by the definition of V_{κ}). The input $\operatorname{Ind}[e]^{\oplus v} \in D_G$ is similarly consistent with $\pi|_{\oplus\kappa}$. Therefore, each $v \in V_{\kappa}$ is a distinct relevant variable for IsEdge $_{\pi|_{\oplus\kappa}}$ over D_G :

$$\operatorname{IsEdge}_{\pi|_{\oplus\kappa}}(\operatorname{Ind}[e]) = 1 \text{ and } \operatorname{IsEdge}_{\pi|_{\oplus\kappa}}(\operatorname{Ind}[e]^{\oplus v}) = 0$$

It follows that $\operatorname{Rel}(\operatorname{IsEdge}_{\pi|_{\oplus_{\kappa}}}; D_G) \geq |V_{\kappa}| = |E_{\kappa}|$ as desired.

Proof of Claim 6.3. Let ρ be a certificate for IsEdge over D_G on 0^n . By Proposition 6.4, the variables of ρ form a vertex cover and so $|\rho| \geq k'$ where k' is the size of a vertex cover of G. Let $\pi = (\overline{v}_{i_1}, \ldots, \overline{v}_{i'_{\mu}}) \in \text{Perm}(\rho)$ be an arbitrary permutation of ρ . Then:

=

$$\sum_{\kappa=1}^{|\pi|} \operatorname{Rel}(\operatorname{IsEdge}_{\pi|\oplus j}; D_G) \ge \sum_{\kappa=1}^{|\pi|} |E(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}})|$$
(Lemma 6.5)

$$m.$$
 (Fact 5.1)

It follows from Theorem 6 that $DT(IsEdge, D_G) \ge k' + m$.

6.3 Hardness distillation for *l*-IsEdge

Following the ideas from Section 6.2, we show that the following set of inputs forms a coreset of ℓ-IsEdge.

Definition 7 (Coreset for ℓ -ISEDGE). For an n-vertex, m-edge graph G and $\ell \in \mathbb{N}$, the set ℓ -D_G \subseteq $\{0,1\}^n \times (\{0,1\}^\ell)^n$ consists of the $m + m(2\ell+2) + 1$ many points

- all generalized edge indicators: ℓ -Ind $[e] \in (\{0,1\}^n)^{\ell+1}$ for each edge $e \in E$ where ℓ -Ind[e] := $(\text{Ind}[e])^{\ell+1};$
- 1-coordinate perturbations of edge indicators: $2\ell + 2$ many points for each $e \in E$ obtained by flipping one of the 1-coordinates in ℓ -Ind[e]; and
- the all 0s input: $0^{n\ell+n}$

Example. See Figure 8 for an example of a graph G and the associated set of inputs ℓ -D_G.



Figure 8: Example of a graph G on four vertices and the associated set of inputs ℓ -D_G with $\ell = 2$. The colored collection of points correspond to edge indicators. The next collection of points correspond to 1-coordinate perturbations of the duplicated variables of the edge indicator for the edge $e = \{v_1, v_2\}$. The perturbed coordinates are bold.

Recall from Theorem 2 that $DT(\ell\text{-IsEdge}) \ge (\ell + 1) \cdot (k' + m)$ where k' is the size of a vertex cover for G. The main claim of this section is that $\ell\text{-}D_G$ distills this hardness factor of $\ell\text{-IsEdge}$.

Claim 6.6 (ℓ - D_G is a coreset for ℓ -IsEdge). Let G be an m-vertex graph and $\ell \in \mathbb{N}$ be arbitrary. Then,

$$DT(\ell$$
-IsEdge, ℓ - $D_G) \ge (\ell + 1)(k' + m)$

where k' is the size of a vertex cover for G.

This claim is analgous to Claim 6.3 and the proof similarly proceeds in two steps. Ultimately, we will apply Theorem 6 where f is ℓ -IsEdge : $\{0,1\}^N \to \{0,1\}, D$ is ℓ - D_G , and x is 0^N . As such, the first step extends Proposition 6.4 to ℓ -IsEdge and shows that certificates for 0^N contain vertex covers. The second step extends Lemma 6.5 and lower bounds the number of relevant variables of subfunctions of ℓ -IsEdge induced by certificates of 0^N .

Proposition 6.7 (Any certificate of 0^N contains a vertex cover). Let G be a graph and let $\{\overline{v}_{i_1}^{(j_1)}, \ldots, \overline{v}_{i_k}^{(j_k)}\}\$ be a certificate for ℓ -IsEdge over ℓ - D_G on 0^N . Then, the vertices $\{v_{i_1}, \ldots, v_{i_k}\}\$ form a vertex cover of G.

Proof. If an edge e is not covered by the vertices $\{v_{i_1}, \ldots, v_{i_k}\}$, then the 1-input ℓ -Ind[e] is consistent with any restriction of the form $\rho = \{\overline{v}_{i_1}^{(j_1)}, \ldots, \overline{v}_{i_k}^{(j_k)}\}$. Therefore, any such ρ cannot be a certificate.

Lemma 6.8 (Lower bounding the number of relevant variables of ℓ -IsEdge subfunctions). Let ρ be a certificate for ℓ -IsEdge over ℓ - D_G on 0^N and $\pi = (\overline{v}_{i_1}^{(j_1)}, \ldots, \overline{v}_{i_k}^{(j_k)}) \in \text{Perm}(\rho)$, a permutation of ρ . Then

$$\operatorname{Rel}(\ell\operatorname{-IsEdge}_{\pi|_{\oplus_{\kappa}}}, \ell \cdot D_G) \ge \ell + (\ell + 1) \cdot |E(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}})|$$

for all $\kappa \in [k]$ such that $E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}}) \neq \emptyset$.

Proof. Let $\pi|_{\oplus\kappa}$ be as in the lemma statement and let E_{κ} and V_{κ} denote $E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ and $V(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$, respectively (recall these quantities from Definitions 3 and 4). If $E_{\kappa} \neq \emptyset$, then we will show that $v_{i_{\kappa}}$ contributes ℓ relevant variables to $\operatorname{Rel}(\ell\operatorname{-IsEdge}_{\pi|_{\oplus\kappa}}, \ell \cdot D_G)$ and that each $v \in V_{\kappa}$ contributes $\ell + 1$.

The vertex $v_{i_{\kappa}}$ contributes ℓ relevant variables. By assumption, E_{κ} is nonempty so there is some edge $e = (v_{i_{\kappa}}, v) \in E_{\kappa}$. The restriction $\pi|_{\oplus\kappa}$ sets one coordinate, $v_{i_{\kappa}}^{j_{\kappa}}$, to 1 and the other coordinates: $\{v_{i_{1}}, \ldots, v_{i_{\kappa-1}}\}$ are set to 0. Since $v \notin \{v_{i_{1}}, \ldots, v_{i_{\kappa-1}}\}$, the input ℓ -Ind $[e] \in \{0, 1\}^{N}$ is consistent with $\pi|_{\oplus\kappa}$. All of the coordinates in $\text{Dup}(v_{i_{\kappa}}) \cup \{v_{i_{\kappa}}^{(0)}\}$ are set to 1 in the input ℓ -Ind[e]. Hence, for any $v' \in \text{Dup}(v_{i_{\kappa}}) \cup \{v_{i_{\kappa}}^{(0)}\} \setminus \{v_{i_{\kappa}}^{(j_{\kappa})}\}$, the input ℓ -Ind $[e]^{\oplus v'}$ is consistent with $\pi|_{\oplus\kappa}$ since $v' \neq v_{i_{\kappa}}^{(j_{\kappa})}$. Therefore,

$$\ell$$
-IsEdge _{$\pi|_{\oplus_{\kappa}}$} (ℓ -Ind[e]) = 1 and ℓ -IsEdge _{$\pi|_{\oplus_{\kappa}}$} (ℓ -Ind[e] ^{$\oplus_{v'}$}) = 0

and ℓ -Ind $[e], \ell$ -Ind $[e]^{\oplus v'} \in \ell$ - D_G . Since v' was arbitrary this shows that each of the ℓ variables in $\operatorname{Dup}(v_{i_{\kappa}}) \cup \{v_{i_{\kappa}}^{(0)}\} \setminus \{v_{i_{\kappa}}^{(j_{\kappa})}\}$ is relevant for ℓ -IsEdge $_{\pi|_{\oplus\kappa}}$ over ℓ - D_G .

Each vertex $v \in V_{\kappa}$ contributes $\ell + 1$ relevant variables. Let $v \in V_{\kappa}$ be an arbitrary vertex and let $e = (v_{i_{\kappa}}, v) \in E_{\kappa}$ be the edge incident to $v_{i_{\kappa}}$ that contains v. Let $v' \in \text{Dup}(v) \cup \{v^{(0)}\}$ be a coordinate of ℓ -IsEdge. As above, the inputs ℓ -Ind[e] and ℓ -Ind $[e]^{\oplus v'}$ are both consistent with the restriction $\pi|_{\oplus\kappa}$. Moreover,

$$\ell$$
-IsEdge _{$\pi|_{\oplus\kappa}$} (ℓ -Ind[e]) = 1 and ℓ -IsEdge _{$\pi|_{\oplus\kappa}$} (ℓ -Ind[e] ^{$\oplus v'$}) = 0

and ℓ -Ind $[e], \ell$ -Ind $[e]^{\oplus v'} \in \ell$ - D_G . This shows that all $\ell + 1$ variables in $\text{Dup}(v) \cup \{v^{(0)}\}$ for $v \in V_{\kappa}$ is relevant. All of these relevant variables are unique and so the total number of relevant variables of ℓ -IsEdge_{$\pi|_{\oplus\kappa}$} is at least $\ell + (\ell + 1)|_{V_{\kappa}}| = \ell + (\ell + 1)|_{E_{\kappa}}|$ as desired.

Proof of Claim 6.6. Let ρ be a certificate for ℓ -IsEdge over ℓ - D_G on 0^N . By Proposition 6.7, the variables in ρ form a vertex cover and so $|\rho| \geq$ the size of a vertex cover of G. Let $\pi = (\overline{v}_{i_1}^{(j_1)}, \ldots, \overline{v}_{i_k}^{(j_k)}) \in \operatorname{Perm}(\rho)$ be a permutation of ρ . In order to apply hardness distillation (Theorem 6), we need to lower bound the number of relevant variables of ℓ -IsEdge_{$\pi|\oplus\kappa$} for $\kappa = 1, 2, \ldots, |\pi|$. However, the lower bound from Lemma 6.8 only applies if the corresponding restricted edge neighborhood $E_{\kappa} = E(v_{i_{\kappa}}; v_{i_1}, \ldots, v_{i_{\kappa-1}})$ is nonempty. To this end, we consider the restriction $\rho' = \{\overline{v}_{i_{\kappa}}^{(j_{\kappa})} \mid E_{\kappa} \neq \emptyset\} \subseteq \rho$. This restriction is still a certificate for ℓ -IsEdge over ℓ - D_G on 0^N and therefore must still contain a vertex cover by Proposition 6.7. Therefore, $|\rho'| \geq k'$ where k' is the size of a vertex cover of G. We can now write

$$\sum_{\kappa=1}^{|\pi|} \operatorname{Rel}(\ell\operatorname{-IsEdge}; \ell - D_G) \ge \sum_{\substack{\kappa \in [|\pi|] \\ E_{\kappa} \neq \emptyset}} \operatorname{Rel}(\ell\operatorname{-IsEdge}_{\pi|_{\bigoplus_{\kappa}}}; \ell - D_G)$$

$$\ge \sum_{\substack{\kappa \in [|\pi|] \\ E_{\kappa} \neq \emptyset}} \ell + (\ell + 1)|E_{\kappa}| \qquad (\text{Lemma 6.8})$$

$$= |\rho'|\ell + (\ell + 1)m \qquad (\text{Fact 5.1: } \{E_{\kappa}\} \text{ partition } E)$$

$$\ge \ell k' + (\ell + 1)m. \qquad (\rho' \text{ contains a vertex cover})$$

We have satisfied the conditions of Theorem 6 with f being ℓ -IsEdge, D being ℓ - D_G , and x being 0^N . We conclude

$$DT(\ell\text{-IsEdge}, \ell\text{-}D_G) \ge k' + k'\ell + (\ell+1)m = (\ell+1)(k'+m).$$

6.4 Learning consequence for inverse polynomial error

In this section, we use Claim 6.6 to obtain hardness of learning decision trees with membership queries. We recall, formally, the learning problem we are interested in.

DT-LEARN (n, s, s', ε) : Given random examples from an unknown distribution \mathcal{D} and membership queries to a size-s target decision tree, output a size-s' decision tree which ε -approximates the target over \mathcal{D} .

Theorem 7 (Hardness learning DTs with inverse polynomial error). For all constants $\delta' > 0, d \in \mathbb{N}$, there is a sufficiently small constant $\delta > 0$ such that the following holds. If DT-LEARN $(n, s, (1 + \delta) \cdot s, \varepsilon)$ with $s = O(n), \varepsilon = O(1/n)$ can be solved in randomized time t(n), then VERTEXCOVER $(k, (1 + \delta') \cdot k)$ on degree-d, n-vertex graphs can be solved in randomized time $O(n^2 \cdot t(n^2))$.

Proof. Given $\delta' > 1$ and $d \in \mathbb{N}$, let $\lambda < 1$ be any large enough constant so that $\lambda(1 + \delta') > 1$ and let $\delta > 0$ be any constant satisfying $1 < (1 + \delta) < \min\{\lambda(1 + \delta'), 1 + \frac{1-\lambda}{d}\}$. Then we will use a learner for DT-LEARN $(n, s, (1 + \delta) \cdot s, \varepsilon)$ to solve VERTEXCOVER $(k, (1 + \delta')k)$.

The reduction. Fix $\ell = \Theta(n)$ large enough so that $1 + \frac{1-\lambda}{d} > (1+\delta) + \frac{2(1+\delta)n}{\ell}$. Such an ℓ exists since $1 + \frac{1-\lambda}{d} > 1 + \delta$ by assumption. Consider the function ℓ -IsEdge : $\{0, 1\}^N \to \{0, 1\}$ and the set of inputs ℓ - D_G for $N = n + \ell n = \Theta(n^2)$. Let \mathcal{D} be the distribution which is uniform over the set ℓ - D_G and fix $\varepsilon < 1/|\text{supp}(\ell - D_G)| = O(1/m^2)$ (which is O(1/N) since $n = \Theta(m)$ for constant degree graphs) and $s = \ell(k+m) + 2mn = O(N)$. Run the procedure in Figure 9.

VERTEXCOVER $(k, (1 + \delta') \cdot k)$:

Given: G, an m-edge degree-d graph over n vertices and $k \in \mathbb{N}$

Run: DT-LEARN $(N, s, (1 + \delta) \cdot s, \varepsilon)$ for t(N) time steps providing the learner with

- queries: return ℓ -IsEdge $(v^{(0)}, \ldots, v^{(\ell)})$ for a query $(v^{(0)}, \ldots, v^{(\ell)}) \in \{0, 1\}^N$; and
- random samples: return $(\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(\ell)}) \sim \mathcal{D}$ for a random sample.

 $T_{\text{hyp}} \leftarrow \text{decision tree output of the learner}$

 $\varepsilon_{\text{hyp}} \leftarrow \text{dist}_{\mathcal{D}}(T_{\text{hyp}}, \ell\text{-IsEdge})$

Output: YES if and only if $|T_{hyp}| \leq (1+\delta) \cdot [\ell(k+m) + 2mn]$ and $\varepsilon_{hyp} \leq \varepsilon$

Figure 9: Using an algorithm for DT-LEARN to solve VERTEXCOVER.

Runtime. Any query $(v^{(0)}, \ldots, v^{(\ell)}) \in \{0, 1\}^N$ to ℓ -IsEdge can be answered in O(N) time by looking at G and computing IsEdge $(v^{(0)})$ in time O(m) then checking that the appropriate vertices are set to 1. Similarly, a random sample from \mathcal{D} can be obtained in time O(N) by picking a uniform random element of ℓ - D_G . This algorithm for VERTEXCOVER requires $O(N \cdot t(N))$ time to run the learner plus time $O(N^2)$ to compute dist $_{\mathcal{D}}(T_{\text{hyp}}, \ell$ -IsEdge). Since $t(N) \geq N$, this implies an overall runtime of $O(N \cdot t(N))$ which is $O(n^2 \cdot t(n^2))$.

Correctness. For correctness, we analyze the yes and no cases separately.

Yes case: $VC(G) \le k$. In this case, Theorem 2 ensures that

$$DT(\ell\text{-IsEdge}) \le (\ell+1)(k+m) + mn \le \ell(k+m) + 2mn.$$

Therefore after t(N) time steps, with high probability, the learner outputs a decision tree T_{hyp} satisfying

$$\operatorname{dist}_{\mathcal{D}}(T_{\operatorname{hyp}}, \ell\operatorname{-IsEdge}) \leq \epsilon$$

and

$$\begin{aligned} |T_{\text{hyp}}| &\leq (1+\delta) \cdot \text{DT}(\ell\text{-IsEdge}) & (\text{learner assumption}) \\ &\leq (1+\delta) \cdot [\ell(k+m)+2mn] & (\text{Theorem 2}) \end{aligned}$$

which ensures that our algorithm correctly outputs YES.

No case: $VC(G) > (1 + \delta') \cdot k$. Assume that $dist_{\mathcal{D}}(T_{hyp}, \ell\text{-IsEdge}) \leq \varepsilon < 1/|supp(\ell - D_G)|$ (otherwise the algorithm correctly outputs NO). In particular, $dist_{\mathcal{D}}(T_{hyp}, \ell\text{-IsEdge}) = 0$. We would like to show that, under our assumption on VC(G), $|T_{hyp}| > (1 + \delta) \cdot [\ell(k + m) + 2mn]$. We start by bounding the vertex cover size of G:

$$\begin{aligned} \operatorname{VC}(G) &= \lambda \operatorname{VC}(G) + \frac{1-\lambda}{d} \operatorname{dVC}(G) \\ &\geq \lambda \operatorname{VC}(G) + \frac{1-\lambda}{d} m \end{aligned} \qquad (Fact 4.1) \\ &\geq \lambda \operatorname{VC}(G) + \left(\delta + \frac{2(1+\delta)n}{\ell}\right) m \qquad (\frac{1-\lambda}{d} > \delta + \frac{2(1+\delta)n}{\ell}) \\ &> (1+\delta)k + \left(\delta + \frac{2(1+\delta)n}{\ell}\right) m. \qquad (\lambda \operatorname{VC}(G) > \lambda(1+\delta')k > (1+\delta)k) \end{aligned}$$

This implies that

$$\operatorname{VC}(G) > (1+\delta)\ell k + \delta\ell m + 2(1+\delta)mn.$$
(1)

We can now write

$$\begin{aligned} |T_{\text{hyp}}| &\geq \text{DT}(\ell\text{-IsEdge}, \ell\text{-}D_G) & (T_{\text{hyp}} \text{ computes } \ell\text{-IsEdge over } \ell\text{-}D_G) \\ &> \ell(\text{VC}(G) + m) & (\text{Claim 6.6}) \\ &> (1 + \delta)\ell k + (1 + \delta)\ell m + 2(1 + \delta)mn & (\text{Equation (1)}) \\ &= (1 + \delta) \cdot [\ell(k + m) + 2mn] \end{aligned}$$

which ensures that our algorithm correctly outputs No.

ľ

7 Hardness for constant error

7.1 Hardness of partial vertex cover

Partial vertex cover. For a graph G = (V, E) and $\alpha \in [0, 1)$, an α -partial vertex cover is subset of the vertices $C \subseteq V$ such that C covers at least a $(1 - \alpha)$ -fraction of the edges. The problem 0-partial vertex cover is the ordinary vertex cover problem. We write $VC_{\alpha}(G) \in \mathbb{N}$ to denote the size of the smallest α -partial vertex cover of G. See Figure 10 for an example of a partial vertex cover. The problem α -partial (k, k')-VERTEXCOVER is to distinguish whether there exists an α partial vertex cover of size $\leq k$ or every α -partial vertex cover requires size > k'. As with ordinary vertex cover, solving this gapped problem is equivalent to approximating α -partial vertex cover. Theorem 4 implies hardness of approximating α -partial vertex cover. It is possible to upgrade an α -partial vertex cover to an ordinary vertex cover by augmenting it with the vertices of uncovered edges.

Fact 7.1 (Upgrading α -partial vertex covers). Any α -partial vertex cover C for a graph G with m-edges can be transformed into a vertex cover C' for G satisfying $|C'| \leq |C| + 2\alpha m$.



(a) A vertex cover and its covered edges highlighted in teal (b) A $\frac{1}{5}$ -partial vertex cover and its covered edges highlighted in purple

Figure 10: A graph G = (V, E) with 10 edges having VC(G) = 3 and $VC_{1/5}(G) = 2$.

By definition, if C is an α -partial vertex cover, then C leaves at most αm edges uncovered. Augmenting C with the $\leq 2\alpha m$ vertices of these uncovered edges yields a vertex cover of G. The size of the resulting vertex cover is $\Theta(m)$ which would be problematic if G has small vertex covers. Fortunately, for constant degree graphs, $VC(G) = \Theta(m)$ (Fact 4.1), so α -partial vertex covers for these graphs are close to optimal vertex covers. This enables us to show that α -partial vertex cover on constant degree graphs is just as hard to approximate as vertex cover. Claim 2.2 follows by combining Claim 7.2 with Theorem 4.

Claim 7.2 (Hardness of approximating α -partial vertex cover). For every constant c' > 1 and $d \in \mathbb{N}$, there are constants $\alpha \in (0, 1)$ and c > 1 such that if there is an algorithm solving α -partial $(k, c \cdot k)$ -VERTEXCOVER on n-vertex, degree-d graphs in time t(n), then there is an algorithm for solving $(k, c' \cdot k)$ -VERTEXCOVER on n-vertex degree-d graphs in time t(n). One can assume that $\alpha < \frac{1}{d+1}$.

Proof. Given c' > 1, let $\alpha \in (0,1)$ be small enough so that $1 < (1 - 2\alpha d)c'$ (and also small enough so that $\alpha < \frac{1}{d+1}$ for the second part of the claim) and let c be any constant satisfying $1 < c < (1 - 2\alpha d)c'$. We will solve $(k, c' \cdot k)$ -VERTEXCOVER using an algorithm for α -partial $(k, c \cdot k)$ -VERTEXCOVER.

Given a graph G and a parameter k, run the algorithm for α -partial $(k, c \cdot k)$ -VERTEXCOVER on G and k. Output YES if and only if the algorithm returns YES. We claim that this procedure solves $(k, c' \cdot k)$ -VERTEXCOVER on degree-d graphs. For correctness, we analyze the yes and no cases separately.

Yes case: $VC(G) \le k$. In this case, we have

$$\operatorname{VC}_{\alpha}(G) \leq \operatorname{VC}(G) \leq k$$

and so the algorithm correctly outputs YES.

No case: VC(G) > c'k. Let *m* denote the number of edges of *G* and let *C* be the smallest α -partial vertex cover of *G*. Fact 7.1 implies that $|C'| - 2\alpha m \le |C| = VC_{\alpha}(G)$ where *C'* is a (possibly suboptimal) vertex cover for *G*. Therefore,

$$\begin{aligned} \operatorname{VC}_{\alpha}(G) &\geq \operatorname{VC}(G) - 2\alpha m & (Fact 7.1) \\ &\geq \operatorname{VC}(G) - 2\alpha d \cdot \operatorname{VC}(G) & (Fact 4.1) \\ &> (1 - 2\alpha d)c'k & (\operatorname{VC}(G) > ck \text{ by assumption}) \\ &> ck & (c < (1 - 2\alpha d)c') \end{aligned}$$

which means the algorithm correctly outputs No.

7.2 Definition of the hard distribution

For Theorem 7, we used the distribution which was uniform over the set ℓ - D_G . This distribution has the property that the target function ℓ -IsEdge *can* be approximated with subconstant error by a small decision tree. In fact, the constant function f(x) = 0 obtains error $\leq \Pr[\ell$ -IsEdge = 1] $\leq 1/m$ in approximating ℓ -IsEdge. Therefore, to obtain hardness in the constant-error regime, we need to define a new distribution, one over which the target function ℓ -IsEdge is close to balanced. To this end, we define the following distribution.

Definition 8 (Constant-error hard distribution). For a graph G and $\ell \in \mathbb{N}$, the distribution ℓ - \mathcal{D}_G over $\{0,1\}^n \times (\{0,1\}^\ell)^n$ is obtained via the following experiment

- with probability 1/2 sample the all 0s input;
- with probability 1/4 sample a generalized edge indicator, ℓ -Ind[e] for $e \in E$ uniformly at random;
- with probability 1/4 sample a 1-coordinate perturbation of an edge indicator uniformly at random.

We prove the following analogue of Claim 6.6 which shows that constant error decision trees must have large size.

Claim 7.3. Let G be an m-edge graph, $\ell \in \mathbb{N}$, and $\alpha \in (0,1)$. If T is a decision tree satisfying

$$\operatorname{dist}_{\ell - \mathcal{D}_G}(T, \ell \operatorname{-IsEdge}) \leq \frac{1}{16} \cdot \alpha$$

then

$$|T| \ge (\ell + 1) \cdot \left[\operatorname{VC}_{\alpha}(G) + (1 - \alpha)m \right].$$

We first need the following lemma showing how to extract an α -partial vertex cover from a decision tree for ℓ -IsEdge.

Lemma 7.4 (Obtaining an α -partial vertex cover from a constant-error decision tree for ℓ -IsEdge). Let T be a decision tree satisfying

$$\operatorname{dist}_{\ell - \mathcal{D}_G}(T, \ell \operatorname{-IsEdge}) < \frac{1}{4}\alpha$$

for any constant $\alpha \in (0,1)$. Then:

- 1. the set $E' = \{e \in E \mid T(\ell \operatorname{-Ind}[e]) = 1\}$ satisfies $|E'| \ge (1-a)m$; and
- 2. if $\pi = (\overline{v}_{i_1}^{(j_1)}, \dots, \overline{v}_{i_k}^{(j_k)})$ is the path followed by 0^N in T, then for $E_{\kappa} = E(v_{i_{\kappa}}; v_{i_1}, \dots, v_{i_{\kappa-1}})$, the set of vertices

$$C = \{ v_{i_{\kappa}} \mid E' \cap E_{\kappa} \neq \emptyset \}$$

covers all edges in E'. In particular, C is an α -partial vertex cover.

Proof. We prove the two points separately.

First point: $|E'| \ge (1 - \alpha)m$. The set of edges $E \setminus E'$ correspond to inputs ℓ -Ind[e] such that $T(\ell$ -Ind[e]) = 0 but ℓ -IsEdge(ℓ -Ind[e]) = 1. Since each input ℓ -Ind[e] has mass $\frac{1}{4m}$ over ℓ - \mathcal{D}_G , we have

$$\frac{1}{4}\alpha > \operatorname{dist}_{\ell - \mathcal{D}_G}(T, \ell \text{-IsEdge}) \qquad (Assumption) \\
\geq |E \setminus E'| \cdot \frac{1}{4m} \qquad (Definition of E') \\
= (m - |E'|) \cdot \frac{1}{4m}.$$

Second point: C is an α -partial vertex cover. Since dist $(T, \ell$ -IsEdge) < 1/2, we know that $T(0^N) = 0$ and therefore the path π terminates in a 0-leaf. For every edge $e \in E'$, the input ℓ -Ind[e] must diverge from π at some point $v_{i_{\kappa}}^{j_{\kappa}}$. This κ then satisfies $e \in E_{\kappa}$ so that $e \in E' \cap E_{\kappa} \neq \emptyset$. It follows that C covers the edge e. Since E' constitutes at least a $(1 - \alpha)$ -fraction of the edges, C is an α -partial vertex cover.

Proof of Claim 7.3. Let T be any decision tree such that

$$\operatorname{dist}_{\ell - \mathcal{D}_G}(T, \ell \operatorname{-IsEdge}) \leq \frac{1}{16} \alpha.$$

In particular, T satisfies the conditions of Lemma 7.4. Let E', π, E_{κ} , and C be as in the statement of Lemma 7.4. For each $v_{i_{\kappa}} \in C$, we define

$$R(v_{i_{\kappa}}) \coloneqq \mathrm{DUP}(v_{i_{\kappa}}) \cup \left\{ v_{i_{\kappa}}^{(0)} \right\} \setminus \left\{ v_{i_{\kappa}}^{(j_{\kappa})} \right\} \cup \bigcup_{\{v_{i_{\kappa}}, v\} \in E' \cap E_{\kappa}} \mathrm{DUP}(v) \cup \left\{ v^{(0)} \right\}.$$

Furthermore, let T_{κ} be the subtree which is the right child of $\pi(\kappa)$. That is T_{κ} is the subtree of T which catches all of the inputs ℓ -Ind[e] for $e \in E' \cap E_{\kappa}$. Recall from the proof of Lemma 6.8

that the variables in $R(v_{i_{\kappa}})$ are all relevant for the subfunction ℓ -IsEdge $_{\pi|\oplus\kappa}$. Each such relevant variable which is *not* queried in the subtree T_{κ} results in an error. For example, if T_{κ} does not query a variable $v' \in \text{Dup}(v_{i_{\kappa}}) \cup \left\{ v_{i_{\kappa}}^{(0)} \right\} \setminus \left\{ v_{i_{\kappa}}^{(j_{\kappa})} \right\}$, then the string ℓ -Ind $[e]^{\oplus v'}$ where $e \in E' \cap E_{\kappa}$ is classified as 1 by T:

$$T(\ell\operatorname{-Ind}[e]^{\oplus v'}) = T_{\kappa}(\ell\operatorname{-Ind}[e]^{\oplus v'}) = T_{\kappa}(\ell\operatorname{-Ind}[e]) = 1$$

whereas ℓ -IsEdge $(\ell$ -Ind $[e]^{\oplus v'}) = 0$. Thus, each relevant variable which is *not* queried in the subtree T_{κ} results in a 0-input being misclassified as 1. Each such misclassification contributes $\Pr_{\ell \mathcal{D}_G}[\ell$ -Ind $[e]^{\oplus v'}] \geq \frac{1}{4} \cdot \frac{1}{m(2\ell+2)}$ to dist_{\ell \mathcal{D}_G}(T, \ell-IsEdge). Therefore, we can write

$$\frac{1}{16} \alpha \geq \operatorname{dist}_{\ell \cdot \mathcal{D}_{G}}(T, \ell \operatorname{-IsEdge})$$

$$\geq \left(\sum_{v_{i_{\kappa}} \in C} |R(v_{i_{\kappa}})| - |T_{\kappa}|\right) \cdot \frac{1}{8(m\ell + m)} + (m - |E'|) \cdot \frac{1}{4m}$$

$$\geq \left(\sum_{v_{i_{\kappa}} \in C} |R(v_{i_{\kappa}})| - |T_{\kappa}|\right) \cdot \frac{1}{16\ell m} + (m - |E'|) \cdot \frac{1}{4m} \qquad (m \leq \ell m)$$

where the quantity $\sum_{v_{i_{\kappa}} \in C} |R(v_{i_{\kappa}})| - |T_{\kappa}|$ counts how many 0-inputs are misclassified as 1 by T and m - |E'| counts how many 1-inputs are misclassified as 0. These quantities are weighted by the respective masses of each type of input over $\ell - \mathcal{D}_G$. Rearranging gives the lower bound:

$$\begin{split} \sum_{v_{i_{\kappa}} \in C} |T_{\kappa}| &\geq 4\ell(m - |E'|) - \alpha\ell m + \sum_{v_{i_{\kappa}} \in C} |R(v_{i_{\kappa}})| \\ &\geq 4\ell(m - |E'|) - \alpha\ell m + \ell|C| + \sum_{v_{i_{\kappa}} \in C} (\ell + 1)|E' \cap E_{\kappa}| \qquad (|\operatorname{DuP}(v) = \ell|) \\ &= 4\ell(m - |E'|) - \alpha\ell m + \ell|C| + (\ell + 1)|E'| \qquad (\{E' \cap E_{\kappa}\} \text{ partitions } E') \\ &\geq 4\ell(m - |E'|) - \alpha\ell m + \operatorname{VC}_{\alpha}(G)\ell + (\ell + 1)|E'| \qquad (C \text{ is an } \alpha \text{-partial vertex cover}) \\ &\geq \ell \operatorname{VC}_{\alpha}(G) + (1 - \alpha)(\ell + 1)m. \qquad (|E'| \leq m) \end{split}$$

Therefore, since the T_{κ} and π are all disjoint parts of T:

$$T| \geq |\pi| + \sum_{v_{i_{\kappa}} \in C} |T_{\kappa}|$$

$$\geq |C| + \sum_{v_{i_{\kappa}} \in C} |T_{\kappa}|$$
 (Definition of C)

$$\geq \operatorname{VC}_{\alpha}(G) + \sum_{v_{i_{\kappa}} \in C} |T_{\kappa}|$$
 (C is an α -partial vertex cover)

$$\geq (\ell + 1) [\operatorname{VC}_{\alpha}(G) + (1 - \alpha)m]$$

which completes the proof.

7.3 Learning consequence for constant-error: Proof of Theorem 1

Theorem 8 (Hardness of learning DTs with constant error). For all constants $\delta' > 0$, $d \in \mathbb{N}$, and $\alpha < \frac{1}{d+1}$, there is a sufficiently small constant $\delta > 0$ such that the following holds. If DT-LEARN $(n, s, (1 + \delta) \cdot s, \varepsilon)$ with s = O(n) and $\varepsilon = \Theta(1)$ can be solved in randomized time t(n), then α -PARTIALVERTEXCOVER $(k, (1 + \delta')k)$ on degree-d graphs can be solved in time $O(n^2t(n^2))$.

The proof of this theorem is similar to that of Theorem 7. The main difference is that our lower bound on the decision tree size of ℓ -IsEdge in the constant-error regime is quantitatively weaker than that of Claim 6.6. We will need to make the appropriate adjustments to the approximation factor of the DT-LEARNER in order to tolerate the weaker lower bound.

Proof. Let $\delta' > 0$, $d \in \mathbb{N}$, and $\alpha < \frac{1}{d+1}$ be given. The assumption that $\alpha < \frac{1}{d+1}$ implies $\alpha < \frac{1-\alpha}{d}$. Therefore, we can fix some $\lambda < 1$ large enough so that $\lambda(1 + \delta') > 1$ and $\alpha < \frac{(1-\lambda)(1-\alpha)}{d}$. Let $\delta > 0$ be any constant satisfying $\frac{(1-\lambda)(1-\alpha)}{d} > \delta + \alpha$ and $(1 + \delta) < \lambda(1 + \delta')$. We will solve α -PARTIALVERTEXCOVER $(k, (1 + \delta')k)$ using an algorithm for DT-LEARN $(n, s, (1 + \delta) \cdot s, \varepsilon)$.

The reduction. Fix $\ell = \Theta(n)$ large enough so that

$$\frac{(1-\lambda)(1-\alpha)}{d} > \delta + \alpha + \frac{2(1+\delta)n}{\ell}.$$
(2)

Such an ℓ exists by our assumption that $\frac{1-\lambda}{d} > \delta + \alpha$. As in Theorem 7 our target function will be ℓ -IsEdge : $\{0,1\}^N \to \{0,1\}$ for $N = n + \ell n = \Theta(n^2)$. Our distribution will be ℓ - \mathcal{D}_G and we fix $\varepsilon < \frac{1}{16}\alpha = \Theta(1)$ and $s = \ell(k+m) + 2mn = O(N)$. Run the same procedure as in Figure 9 where the distribution \mathcal{D} is ℓ - \mathcal{D}_G .

Runtime. As in the proof of Theorem 7, queries and random samples for ℓ -IsEdge can be handled in O(N) time. Thus running the learner requires $O(N \cdot t(N))$ time. Computing the error $\operatorname{dist}_{\ell \sim \mathcal{D}_G}(T_{\operatorname{hyp}}, \ell$ -IsEdge) takes $O(N^2)$ time. The overall runtime is therefore $O(N \cdot t(N))$ which is $O(n^2 \cdot t(n^2))$.

Correctness. We analyze the YES case and NO case separately.

Yes case: $VC_{\alpha}(G) \leq k$. This case is identical to the YES case in Theorem 7. So our algorithm correctly outputs YES.

No case: $\operatorname{VC}_{\alpha}(G) > (1 + \delta')k$. Assume that $\operatorname{dist}_{\ell \cdot \mathcal{D}_G}(T_{\operatorname{hyp}}, \ell \operatorname{-IsEdge}) \leq \varepsilon < \frac{1}{16}\alpha$ (otherwise our algorithm correctly outputs NO). We would like to show that $|T_{\operatorname{hyp}}| > (1 + \delta) \cdot [\ell(k + m) + 2mn]$.

We start by bounding α -partial vertex cover size of G:

$$VC_{\alpha}(G) = \lambda VC_{\alpha}(G) + \frac{1-\lambda}{d} dVC_{\alpha}(G)$$

$$\geq \lambda VC_{\alpha}(G) + \frac{(1-\lambda)(1-\alpha)}{d}m \qquad (dVC_{\alpha}(G) \geq (1-\alpha)m \text{ for degree } d \text{ graphs})$$

$$\geq \lambda VC_{\alpha}(G) + \left(\delta + \alpha + \frac{2(1+\delta)n}{\ell}\right)m \qquad (Equation (2))$$

$$\geq (1+\delta)k + \left(\delta + \alpha + \frac{2(1+\delta)n}{\ell}\right)m. \qquad (\lambda VC_{\alpha}(G) > \lambda(1+\delta')k > (1+\delta)k)$$

Rearranging gives

$$\ell \operatorname{VC}_{\alpha}(G) \ge (1+\delta)k\ell + (\delta+\alpha)m\ell + 2(1+\delta)mn.$$
(3)

Therefore,

$$|T_{\text{hyp}}| > \ell(\text{VC}_{\alpha}(G) + (1 - \alpha)m)$$

$$> (1 + \delta)k\ell + (\delta + \alpha)m\ell + 2(1 + \delta)mn + (1 - \alpha)m\ell$$

$$= (1 + \delta) \cdot [\ell(k + m) + 2mn]$$
(Equation (3))

which ensures that our algorithm correctly outputs No.

Remark 3 (Implications for testing decision trees). The above proof of Theorem 8 and the proof of Theorem 7 actually prove hardness of *testing* decision tree size. Specifically, the proof of Theorem 8 shows that any tester which can distinguish whether a target function f is a size-s decision tree or is $\Omega(1)$ -far from every size-s decision tree over a distribution \mathcal{D} can also approximate PAR-TIALVERTEXCOVER. Therefore, the problem of distribution-free testing decision tree size is also NP-hard.

Proof of Theorem 1. If there were an algorithm for learning decision trees which satisfies the constraints of Theorem 1, then Theorem 8 shows that α -PARTIALVERTEXCOVER can be solved in RTIME $(n^2t(n^2))$. Theorem 4 and Claim 7.2 then imply that SAT can be solved in randomized time $O(n^2 \operatorname{polylog} n \cdot t(n^2 \operatorname{polylog} n))$.

Acknowledgments

We thank Pasin Manurangsi for a helpful conversation and the FOCS reviewers for their comments and feedback.

The authors are supported by NSF awards 1942123, 2211237, 2224246 and a Google Research Scholar award. Caleb is also supported by an NDSEG fellowship, and Carmen by a Stanford Computer Science Distinguished Fellowship.

References

[ABF⁺09] Misha Alekhnovich, Mark Braverman, Vitaly Feldman, Adam Klivans, and Toniann Pitassi. The complexity of properly learning simple concept classes. Journal of Computer & System Sciences, 74(1):16–34, 2009. Preliminary version in FOCS 2004. 1.1, 2.1

- [AH12] Micah Adler and Brent Heeringa. Approximating optimal binary decision trees. Algorithmica, 62(3-4):1112–1121, 2012. 1.2
- [ALM⁺98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. J. ACM, 45(3):501–555, may 1998. 4.1
- [Ang] Dana Angluin. Remarks on the difficulty of finding a minimal disjunctive normal form for boolean functions. Unpublished Manuscript. 1, 1.1, 2.1
- [Ang88] Dana Angluin. Queries and concept learning. Machine learning, 2:319–342, 1988. 1.1
- [AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. J. ACM, 45(1):70–122, jan 1998. 4.1
- [BB03] Nader H Bshouty and Lynn Burroughs. On the proper learning of axis-parallel concepts. The Journal of Machine Learning Research, 4:157–176, 2003. 1.2
- [BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989. 1.1
- [BFJ⁺94] Avirm Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning DNF and characterizing statistical query learning using Fourier analysis. In Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC), pages 253–262, 1994. 1.1
- [BKB17] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpretability via model extraction. In Proceedings of the 4th Workshop on Fairness, Accountability, and Transparency in Machine Learning (FAT/ML), 2017. 1
- [BLQT21] Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Decision tree heuristics can fail, even in the smoothed setting. In *Proceedings of the 25th International Conference* on Randomization and Computation (RANDOM), volume 207, pages 45:1–45:16, 2021. 1.2
- [BLQT22] Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Properly learning decision trees in almost polynomial time. Journal of the ACM (JACM), 69(6):39:1–39:19, 2022. 1.1, 3
- [Bre01] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001. 1
- [BS96] Leo Breiman and Nong Shang. Born again trees. Technical report, University of California, Berkeley, 1996. 1
- [Bsh93] Nader Bshouty. Exact learning via the monotone theory. In Proceedings of 34th Annual Symposium on Foundations of Computer Science (FOCS), pages 302–311, 1993. 1, 1.1
- [Bsh23] Nader H. Bshouty. Superpolynomial lower bounds for learning monotone classes. *Electron. Colloquium Comput. Complex.*, TR23-006, 2023. 1.1, 2.1

- [BSS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. 3
- [CPR⁺07] Venkatesan T Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh Mohania. Decision trees for entity identification: Approximation algorithms and hardness results. In Proceedings of the 26th ACM Symposium on Principles of Database Systems (PODS), pages 53–62, 2007. 1.2
- [CS95] Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. Proceedings of the 8th Conference on Advances in Neural Information Processing Systems (NeurIPS), 8:24–30, 1995. 1
- [Din07] Irit Dinur. The PCP theorem by gap amplification. J. ACM, 54(3):12–es, jun 2007. 3
- [EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989. 1.1, 3
- [Fel06] Vitaly Feldman. Hardness of approximate two-level logic minimization and pac learning with membership queries. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 363–372, 2006. 1.2
- [Fel16] Vitaly Feldman. Hardness of proper learning. In Encyclopedia of Algorithms, pages 897–900. 2016. 1
- [FH17] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784, 2017. 1
- [GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998. 1.3
- [GJ79] M. R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979. 1.2
- [GLR99] David Guijarro, Victor Lavin, and Vijay Raghavan. Exact learning when irrelevant variables abound. *Information Processing Letters*, 70(5):233–239, 1999. 1, 1.1, 1.2
- [Hau88] David Haussler. Quantifying inductive bias: AI learning algorithms and valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988. 1.1, 2.1
- [HJLT96] Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, 1996. 1.1, 2.1, 5.3
- [HR76] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is NP-complete. *Information processing letters*, 5(1):15–17, 1976. 1.2
- [KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. SIAM Journal on Computing, 22(6):1331–1348, December 1993. 1.1
- [KPB99] S Rao Kosaraju, Teresa M Przytycka, and Ryan Borgstrom. On an optimal split tree problem. In Workshop on Algorithms and Data Structures, pages 157–168. Springer, 1999. 1.2

- [KST23] Caleb Koch, Carmen Strassle, and Li-Yang Tan. Superpolynomial lower bounds for decision tree learning and testing. In *Proceedings of the 2023 ACM-SIAM Symposium* on Discrete Algorithms (SODA), pages 1962–1994, 2023. 1.1, 2.1
- [Lev73] Leonid A Levin. Universal sorting problem. Problemy Predaci Informacii, 9:265–266, 1973. 1.1, 2.1
- [LN04] Eduardo S Laber and Loana Tito Nogueira. On the hardness of the minimum height decision tree problem. *Discrete Applied Mathematics*, 144(1-2):209–212, 2004. 1.2
- [MR02] Dinesh Mehta and Vijay Raghavan. Decision tree approximations of boolean functions. *Theoretical Computer Science*, 270(1-2):609–623, 2002. 1, 1.1
- [PV88] Leonard Pitt and Leslie G Valiant. Computational limitations on learning from examples. Journal of the ACM (JACM), 35(4):965–984, 1988. 1, 1.1
- [PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. Journal of Computer and System Sciences, 43(3):425–440, 1991.
 4.1, 4.1
- [Rav13] Netanel Raviv. Truth table minimization of computational models. *CoRR*, abs/1306.3766, 2013. 1.2
- [RCC⁺22] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1 – 85, 2022. 1
- [RRV07] Dana Ron, Amir Rosenfeld, and Salil Vadhan. The hardness of the expected decision depth problem. *Information processing letters*, 101(3):112–118, 2007. 1.2
- [Sie08] Detlef Sieling. Minimization of decision trees is hard to approximate. Journal of Computer and System Sciences, 74(3):394–403, 2008. 1.2, 1.3, 2.2.2, 5.3, 5.3, 5, 5.3
- [SS93] Robert Schapire and Linda Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. In *Proceedings of the 6th Annual Conference on Computational Learning Theory (COLT)*, pages 17–26, 1993. 1.1
- [Tre14] Luca Trevisan. Inapproximability of Combinatorial Optimization Problems, chapter 13, pages 381–434. John Wiley & Sons, Ltd, 2014. 4.1
- [VAB07] Anneleen Van Assche and Hendrik Blockeel. Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In *European Conference on Machine Learning (ECML)*, pages 418–429, 2007. 1
- [Val84] Leslie Valiant. A theory of the learnable. Communications of the ACM, 27(11):1134–1142, 1984. 1, 1.1, 1.2
- [Val85] Leslie G Valiant. Learning disjunction of conjunctions. In Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI), pages 560–566, 1985.
 1.2

- [VLJ⁺17] Gilles Vandewiele, Kiani Lannoye, Olivier Janssens, Femke Ongenae, Filip De Turck, and Sofie Van Hoecke. A genetic algorithm for interpretable model extraction from decision tree ensembles. In *Trends and Applications in Knowledge Discovery and Data Mining*, pages 104–115, 2017. 1
- [VS20] Thibaut Vidal and Maximilian Schiffer. Born-again tree ensembles. In Proceedings of the 37th International Conference on Machine Learning (ICML), pages 9743–9753, 2020. 1
- [ZB00] Hans Zantema and Hans Bodlaender. Finding small equivalent decision trees is hard. International Journal of Foundations of Computer Science, 11(2):343–354, 2000. 1.2, 1.3, 2.2.2, 5.3
- [ZH16] Yichen Zhou and Giles Hooker. Interpreting models via single tree approximation, 2016. 1