On Testing Decision Tree

Nader H. Bshouty
Department of Computer Science, Technion, Haifa, Israel

Catherine A. Haddad-Zaknoon
Department of Computer Science, Technion, Haifa, Israel

Abstract

In this paper, we study testing decision tree of size and depth that are significantly smaller than the number of attributes n.

Our main result addresses the problem of $\text{poly}(n, 1/\epsilon)$ time algorithms with $\text{poly}(s, 1/\epsilon)$ query complexity (independent of n) that distinguish between functions that are decision trees of size s from functions that are ϵ-far from any decision tree of size $\phi(s, 1/\epsilon)$, for some function $\phi > s$. The best known result is the recent one that follows from Blanc, Lange and Tan, [3], that gives $\phi(s, 1/\epsilon) = 2^{O((\log^3 s)/\epsilon^3)}$. In this paper, we give a new algorithm that achieves $\phi(s, 1/\epsilon) = 2^{O(\log^2(s/\epsilon))}$.

Moreover, we study the testability of depth-d decision tree and give a *distribution free* tester that distinguishes between depth-d decision tree and functions that are ϵ-far from depth-d^2 decision tree.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases Testing decision trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2022.17

1 Introduction

Decision tree is one of the popular predictive modelling approaches used in many areas including statistics, data mining and machine learning. Recently, property-testing of subclasses of decision trees have attracted much attention [1, 3, 6, 9, 10, 12]. In property testing, the algorithm is provided by an access to a black box to some Boolean function f and labeled random examples of f according to some distribution $\mathcal{D}$. Given a subclass of decision trees C, we need to decide whether f is a decision tree in C or "far" from being in C with respect to $\mathcal{D}$, [4, 11, 13].

Since finding efficient algorithms for this problem is difficult, the following relaxation is considered. Let H be a larger class of decision trees $H \supset C$. Then, we are interested in the question: can we efficiently test C by H? That is, to efficiently decide whether f is a decision tree in C or "far" from being in H with respect to $\mathcal{D}$, [12]. In this context, the challenge is to find a small class $H \supset C$ such that efficient testing algorithm exists. In this paper, we address this problem while examining and constructing algorithms that are efficient in the query complexity and run in polynomial time.

1.1 Models

Let C and $H \supseteq C$ be two classes of Boolean functions $f : \{0,1\}^n \to \{0,1\}$. In the *distribution-free* model, the algorithm has an access to a *black box query* and *random example query*. The black box query, for an input $x \in \{0,1\}^n$ returns $f(x)$. The random example query, when invoked, returns a random example $(x, f(x))$ such that x is chosen according to an arbitrary and unknown distribution $\mathcal{D}$. In the *uniform distribution* model, $\mathcal{D} = U$ is the uniform distribution over $\{0,1\}^n$.

A function $g \in H$ is called ϵ-close to $f \in C$ with respect to the distribution $\mathcal{D}$ if, $\mathbf{Pr}_{\mathcal{D}}[g(x) \neq f(x)] \leq \epsilon$. In the *distribution-free property testing C by H*, [12], (resp. uniform distribution property testing), for *any* Boolean function f, we need to distinguish, with high probability and via the above queries to f, between the case that f is in C versus the case that f is ϵ-far (not ϵ-close) from every function in H with respect to $\mathcal{D}$ (resp. the uniform distribution). Such an algorithm is called a *tester for C by H*. When $H = C$, the tester is called a *tester for C*.

1.2 Decision Tree

A *decision tree* is a rooted binary tree in which each internal node is labeled with a variable x_i and has two children. Each leaf is labeled with an output from $\{0, 1\}$. A decision tree computes a Boolean function in the following way: given an input $x \in \{0, 1\}^n$, the value of the function on x is the output in the leaf reached by the path that starts at the root and goes left or right at each internal node according to whether the variable's value in x is 0 or 1, respectively.

The *size* of a decision tree is the number of leaves of the tree. The *depth* of a node (resp. leaf) in a decision tree is the number of edges in the path from the root to the node (resp. leaf). The depth of the tree is the maximum over all the depth values of its leaves. A *depth-d decision tree T* is a decision tree of depth at most d. A *size-s decision tree T* is a decision tree of size at most s.

1.3 Other Representations of Decision Tree

A *monomial* is a conjunction of variables, and a *term* is a conjunction of literals (variable and negated variable). Two terms t_1 and t_2 are called *disjoint* if $t_1 \wedge t_2 = 0$ (over the field F_2). A *multilinear polynomial* (or just a polynomial) is Boolean function that is defined as a sum of monomials over the field F_2. Every Boolean function can be expressed uniquely as a multilinear polynomial. A *disjoint-terms sum* is a sum of disjoint terms. Every Boolean function can be represented as a disjoint-terms sum. The representation is not unique.

A decision tree f can be represented as a disjoint-terms sum according to the following recurrence. If the decision tree is a leaf, then its disjoint-terms sum representation is the constant in this leaf. If the root label of f is x_i then, $f = x_i f_1 + \overline{x_i} f_0$ where f_1 and f_0 are the disjoint-terms sum of right and left sub-trees of f respectively. It is easy to see that the number of terms we get in this recurrence is equal to the number of leaves in the tree labeled with 1. Therefore, every leaf corresponds to a term and the number of literals in this term is equal to the depth of the leaf. To represent a disjoint-terms sum as a polynomial, we write for each appearance of $\overline{x_i}$ as $x_i + 1$ and expand the expressions with the regular arithmetic rules in the field F_2.

2 Main Result and Technique

In this section, we present previous and new results for testing decision trees. In the next two subsections, we consider two significant results in testing.

2.1 Testing Decision Tree of Size s

Let f be a Boolean function over the variables $x_1, \ldots, x_n$. For $x_{i_1}, \ldots, x_{i_j}$ and $\xi_1, \ldots, \xi_j \in \{0, 1\}$, denote by $f_{|x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j}$ the function that results from substituting $x_{i_r} = \xi_r$, $r = 1, \ldots, j$ in f.

In [3], Blanc, Lange and Tan give the first tester that runs in $\text{poly}(n, 1/\epsilon)$ time with $\text{poly}(\log s,\ 1/\epsilon)\log n$ query complexity and distinguishes between functions that are s-size decision tree and functions that are ϵ-far from every size-$\phi(s, 1/\epsilon)$ decision tree for some function $\phi(s, 1/\epsilon)$. When $n \gg s$, one can use the reduction from [6] to get a $\text{poly}(n, 1/\epsilon)$ time algorithm with $\text{poly}(s, 1/\epsilon)$ query complexity (independent of n) for the same problem. The function ϕ achieved in [3] is $\phi(s, 1/\epsilon) = 2^{(\log^3 s)/\epsilon^3}$. We use a different approach to get $\phi(s, 1/\epsilon) = 2^{\log^2(s/\epsilon)}$.

In fact, Blanc, Lange and Tan [3] solve a more challenging problem. Their tester is a tolerant tester. That is, it distinguishes between functions that are ϵ-close to s-size decision tree and functions that are $\Omega(\epsilon)$-far from every size-$\phi(s, 1/\epsilon)$ decision tree.

To achieve their result, Blanc et. al. define for every function f a complete decision tree $T(d, f)$ of depth $d = O(\log^3 s/\epsilon^3)$ as follows. They define the noise sensitivity of f, $\text{NS}(f)$, as the probability that $f(x) \neq f(x + y)$ where x is uniform random, and for every i, $y_i = 1$ with probability $p = O(\epsilon/\log s)$. The score of f with respect to x_i, $\text{Score}_i(f)$, is defined to be the expected decrease in the noise sensitivity of f provided that x_i is queried. The label of the root of $T(d, f)$ is selected to be the variable x_i that maximizes the score. The left and right sub-trees of $T(d, f)$ are $T(d-1, f_{|x_i\leftarrow 0})$ and $T(d-1, f_{|x_i\leftarrow 1})$, respectively. Then $T(0, g)$ is defined to be a leaf labeled with 0 if $\mathbf{E}[g] < 1/2$ and 1 otherwise. Here g is the function that results from f by substituting the partial assignment defined by the path from the root to the leaf. They prove that, for $d = O(\log^3 s/\epsilon^3)$, if f is a size-s decision tree, then f is $\epsilon/4$-close to $T(d, f)$, and if f is ϵ-far from every size-$2^{O(\log^3 s/\epsilon^3)}$ then, since $T(d, f)$ is size-2^d $\left(= 2^{O(\log^3 s/\epsilon^3)}\right)$ decision tree, f is ϵ-far from $T(d, f)$. Moreover, they show that a query to $T(d, f)$ can be done in $\text{poly}(n, 1/\epsilon)$ time and $\text{poly}(\log s, 1/\epsilon)\log n$ queries. Therefore, $T(d, f)$ and f can be queried to test if they are $\epsilon/4$-close or ϵ-far. By applying the reduction from [6], a tester that solves the same problem in $\text{poly}(n, 1/\epsilon)$ time and $\text{poly}(s, 1/\epsilon)$ queries to f is obtained.

In this paper, we use a different approach. Let f be a size-s decision tree. Our algorithm regards f as a polynomial. Although f may have exponential number of monomials, we are interested in the influential ones only, that is, the small monomials. The number of small monomials in the polynomial representation of size-s decision tree may be exponential. To control the number of small monomials, we shuffle the monomials by choosing a uniform random $a \in \{0, 1\}^n$ and considering $T(x) = f(x + a)$. In disjoint-terms sum representation of f (see Subsection 1.3), a large term t in f (a term that results from a leaf of depth $\Omega(\log(s/\epsilon))$ in the tree), with high probability (w.h.p), more than quarter of its variables become positive in T, and therefore, it only generates large monomials. Therefore, the shuffling process ensures that the number of significant monomials in T is as small as $\text{poly}(s/\epsilon)$. This technique is used in [8] for learning decision tree under the uniform distribution.

Let c be a large constant and let F be the sum of the monomials of size $r = c\log(s/\epsilon)$ in $T(x)$. Let G be the sum of monomials of size greater than r and less than $16r$ in $T(x)$. First, we run the algorithm of Bshouty and Mansour in [8] to exactly learn F and G in polynomial time. If the learning algorithm fails, then w.h.p, f is not size-s decision tree and the algorithm rejects. Then, we define a decision tree $T(d, F, G)$ of depth $d = O(\log^2(s/\epsilon))$ as follows. Define $\text{Frac}(F, x_i)$ to be the fraction of the number of monomials in F that contain x_i. Choose a variable x_{i_1} with the minimum index i_1 that maximizes $\text{Frac}(F, x_{i_1})$ and use it as the label of the root of $T(d, F, G)$. The left and right sub-trees of $T(d, F, G)$ are $T(d-1, F^{(0)}, G^{(0)})$ and $T(d-1, F^{(1)}, G^{(1)})$, respectively, where $F^{(\xi)}$ is the sum of all monomials that appear in $F_{|x_{i_1}\leftarrow\xi}$ and not in $G_{|x_{i_1}\leftarrow\xi}$, and $G^{(\xi)}$ is the sum of all monomials that appear in $G_{|x_{i_1}\leftarrow\xi}$ and not in $F_{|x_{i_1}\leftarrow\xi}$.

We are interested in making a random walk in this tree. Assume we are given the first $m-1$ steps in the random walk $\xi^{(m-1)} = (\xi_1, \xi_2, \ldots, \xi_{m-1}) \in \{0,1\}^{m-1}$, $F^{\xi^{(m-1)}}$ and $G^{\xi^{(m-1)}}$. We find the variable with the minimum index i_m that maximizes $\mathrm{Frac}(F^{\xi^{(m-1)}}, x_{i_m})$. Choose a random uniform $\xi_m \in \{0,1\}$. Then, for $\xi^{(m)} = (\xi_1, \xi_2, \ldots, \xi_m)$, $F^{\xi^{(m)}}$ is defined to be the sum of all the monomials in $F^{\xi^{(m-1)}}_{|x_{i_m} \leftarrow \xi_m}$ that are not in $G^{\xi^{(m-1)}}_{|x_{i_m} \leftarrow \xi_m}$, and $G^{\xi^{(m)}}$ is defined to be the sum of all the monomials in $G^{\xi^{(m-1)}}_{|x_{i_m} \leftarrow \xi_m}$ that are not in $F^{\xi^{(m-1)}}_{|x_{i_m} \leftarrow \xi_m}$. If $F^{\xi^{(m)}}$ is a constant function $\eta \in \{0,1\}$, then we have reached a leaf labeled with η. The way we define $F^{\xi^{(m)}}$ and $G^{\xi^{(m)}}$ turns to be crucial in the algorithm and is needed for its correctness proof. We note here that, unlike the tester of Blanc, Lange and Tan, this tree is not the decision tree of T or F, because every path in the tree treats F and G as sets of monomials rather than functions.

We show that, when f is a size-s decision tree, for a random shuffling and random $\xi = (\xi_1, \xi_2, \ldots, \xi_m) \in \{0,1\}^m$, with high probability, $\mathrm{Frac}(F^{\xi^{(m)}}, x_{i_m})$ is at least $1/O(\log(s/\epsilon))$. Hence, for a random walk in the tree $T(d, F, G)$, each step decreases the number of monomials in $F^{\xi^{(m)}}$ by a factor of $1 - 1/O(\log(s/\epsilon))$ on average. Therefore, since F contains at most $poly(s/\epsilon)$ monomials, with high probability, a random walk in $T(d, F, G)$ reaches a leaf in $O(\log^2(s/\epsilon))$ steps.

Now suppose f is ϵ-far from every size-$2^{O(\log^2(s/\epsilon))}$ decision tree. It might happen that a function f that is ϵ-far from size-$2^{O(\log^2(s/\epsilon))}$ decision tree passes all the above tests, because the above algorithm relies only on the small monomials of T. Moreover, it might happen that the small monomials of such function coincide with the monomials of a small size decision tree. As a result, we add another test at each leaf of the tree that checks if the function T at the leaf of the tree $T_{|x_{i_1} \leftarrow \xi_1, \ldots, x_{i_m} \leftarrow \xi_m}$ is $\epsilon/4$-close to a constant function.

For a function that is ϵ-far from every size-$2^{O(\log^2(s/\epsilon))}$ decision tree, if it is not rejected because its small monomials coincide with the monomial of small size decision tree, then the random walks will often reach a small depth leaf. On the other hand, if almost all the small depth leaves give a good approximation of the function, then T is ϵ-close to a small depth tree. Therefore, the function is rejected with high probability.

We also show that, although the tester treats F and G as sets of monomials and not as functions, if f is size-s decision tree then the tree gives a good approximation of T.

The above tester runs in $poly(n, 1/\epsilon)$ time and queries. Using the reduction in [6], it can be changed to a tester that runs in $poly(s, 1/\epsilon)n$ time and makes $poly(s, 1/\epsilon)$ queries.

2.2 Testing Decision Tree of Depth d

The algorithm of Blanc, Lange and Tan [3] also distinguishes between depth-d decision tree and functions that are ϵ-far from depth-$O(d^3/\epsilon^3)$ decision trees under the uniform distribution. Again, we can make the query complexity independent of n using the reduction of Bshouty, [6], and get a $2^{O(d)}n$ time uniform-distribution tester that asks $2^{O(d)}/\epsilon$ queries and distinguishes between depth-d decision tree and functions that are ϵ-far from depth-$O(d^3/\epsilon^3)$ decision trees.

In this paper, we give a new simple *distribution-free* tester that runs in $2^{O(d)}n$ time, asks $2^{O(d)}/\epsilon$ queries and distinguishes between depth-d decision tree and functions that are ϵ-far from depth-d^2 decision trees.

Our algorithm relies on the following fact. Let f be a depth-d decision tree. Consider the polynomial representation of $f = M_1 + M_2 + \cdots + M_m$. For any maximal monomial $M_i = x_{i_1} x_{i_2} \cdots x_{i_t}$ (a monomial that is not sub-monomial of any other monomial in f) and any $\xi_1, \ldots, \xi_t \in \{0,1\}$, the function $f_{|x_{i_1} \leftarrow \xi_1, \ldots, x_{i_t} \leftarrow \xi_t}$ is depth $(d-1)$-decision tree.

We can define a depth-d^2 decision tree T_f that is equivalent to f as follows: Find a maximal monomial $M_i = x_{i_1}x_{i_2}\cdots x_{i_t}$. Then use all its variables in the first t levels of the decision tree T_f. That is, build a complete tree of depth t that all its nodes at level j are labeled with x_{i_j}. This defines a different path for each $x_{i_1} = \xi_1, \ldots, x_{i_t} = \xi_t$. Then, in the last node of such path, attach the tree T_g for $g = f_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_t}\leftarrow\xi_t}$. Since depth-$d$ decision trees are degree-d polynomials, we have $t \leq d$, and since the decision trees at level t are depth-$(d-1)$ decision trees, the depth of T_f is at most d^2 (in fact it is at most $d(d-1)/2$).

For the tester, we will not construct T_f, but instead, we show that for any assignment a, finding the route that a takes in the tree T_f can be done efficiently. For this end, we first show that if f is a depth-d decision tree then, the relevant variables of f can be found in $\tilde{O}(2^{2d}) + 2^d \log n$ queries. Then, we show that a maximal monomial of f can be found in $\tilde{O}(2^d)$ queries. For an assignment a drawn according to a distribution $\mathcal{D}$, if f is a depth-d decision tree, the route that a takes in T_f ends before depth d^2. If f is ϵ-far from depth-d decision tree, then, either finding the relevant variables of f fails, or finding a maximal monomial of size at most d fails or, with probability at least ϵ, the route in T_f goes beyond depth d^2. The later happens because if it does not for $O(1/\epsilon)$ examples drawn according to a distribution $\mathcal{D}$, then truncating the tree up to depth d^2, results a tree that is w.h.p ϵ-close of a depth d^2-decision tree with respect to $\mathcal{D}$.

Notice that the query complexity of this tester depends on n because finding the relevant variables of f takes $\tilde{O}(2^{2d}) + 2^d \log n$ queries which depends on n. To make the query complexity independent of n, we use the reduction of Bshouty in [6].

2.3 Non-Polynomial Time Testers

A recent breakthrough result of Blanc et. al. [2] with the reduction of Bshouty [6] gives a uniform tester for size-s decision tree that runs in $n(s/\epsilon)^{O(\log((\log s)/\epsilon))}$ time and makes $(s/\epsilon)^{O(\log((\log s)/\epsilon))}$ queries.

3 A Tester for Depth-d Decision Tree

In this section we prove the following result:

▶ **Theorem 1.** *There is a distribution-free tester that makes $q = \tilde{O}(2^{2d}/\epsilon)$ queries to unknown function f, runs in $O(qn)$ time and*

1. *Accepts w.h.p if f is a depth-d decision tree.*
2. *Rejects w.h.p if f is ϵ-far from depth-d^2 decision trees.*

3.1 The Key Lemma

We start with some notations and definitions, and then prove the key Lemma for the tester.

Recall that *monomial* is a conjunction of variables. A k-monomial is a monomial with at most k variables. A polynomial (over the field F_2) is a sum (in the binary field F_2) of monomials. An s-sparse polynomial is a sum of at most s monomials. We say that the polynomial f is of degree-d if its monomials are d-monomials.

We say that x_i is *relevant* variable in f if $f_{|x_i\leftarrow 0} \neq f_{|x_i\leftarrow 1}$. It is well known that (see for example Lemma 4 in [7]):

▶ **Lemma 2.** *A depth-d decision tree is a 3^d-sparse degree-d polynomial with at most 2^d relevant variables.*

Let $f = M_1 + M_2 + \cdots + M_t$ be a polynomial. We say that M_i is a *maximal monomial* of f if M_i is not a sub-monomial of any other monomial M_j, i.e., for every other monomial M_j, there is a variable in M_i that is not in M_j.

We now prove the key Lemma for our tester:

▶ **Lemma 3.** *Let f be a depth-d decision tree and $f = M_1 + M_2 + \cdots + M_t$ be its polynomial representation. Let $M_i = x_{i_1} \cdots x_{i_{d'}}$, $d' \leq d$ be a maximal monomial of f. For any $\xi_1, \xi_2, \ldots, \xi_{d'} \in \{0,1\}$, we have that $f_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$ is depth-$(d-1)$ decision tree.*

Proof. The proof is by induction on the number of variables m of f. The case $m = 1$ is trivial. Now assume that the result holds for any $m \leq k$.

Let T be any depth-d decision tree with $k+1$ variables that represents f. Let $M_i = x_{i_1} \cdots x_{i_{d'}}$, $d' \leq d$ be a maximal monomial of f. Let $X = \{x_{i_1}, \ldots, x_{i_{d'}}\}$. If the variable of the root of T is $x_{i_j} \in X$ then, $T_{|x_{i_j} \leftarrow 0}$ and $T_{|x_{i_j} \leftarrow 1}$ are left and right decision sub-trees of T and are of depth at most $d-1$. Then, $T_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$ is of depth at most $d-1$ for any $\xi_1, \ldots, \xi_{d'} \in \{0,1\}$.

If the variable of the root of the tree T is $x_\ell \notin X$, then the left sub-tree $T_{|x_\ell \leftarrow 0}$ is a depth-$(d-1)$ decision tree and has at most k variables. We now claim that M_i is a maximal monomial of $T_{|x_\ell \leftarrow 0}$. This is because of the fact that substituting $x_\ell = 0$ in the polynomial representation only removes monomials in f. Since x_ℓ is not in M_i, it does not remove M_i. Therefore, M_i is maximal monomial in $T_{|x_\ell \leftarrow 0}$, and by the induction hypothesis $T_{|x_\ell \leftarrow 0, x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$ is depth-$(d-2)$ decision tree.

The right sub-tree $T_{|x_\ell \leftarrow 1}$ is a depth-$(d-1)$ decision tree that has at most k variables. We now claim that $T_{|x_\ell \leftarrow 1}$ also has M_i as a monomial and it is maximal. Assume for the sake of contradiction that M_i is removed or not maximal, then there must be a monomial $M_j = x_\ell M_i$ in T. Since M_i is sub-monomial of M_j, we get a contradiction to the fact that M_i is maximal in f. Therefore, by the induction hypothesis $T_{|x_\ell \leftarrow 1, x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$ is depth-$(d-2)$ decision tree. This implies that

$$T_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}} = x_\ell \cdot T_{|x_\ell \leftarrow 1, x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}} + \overline{x_\ell} \cdot T_{|x_\ell \leftarrow 0, x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$$

is depth-$(d-1)$ decision tree. ◀

For every degree-d polynomial f, we define the following decision tree T_f. If f is constant function, then T_f is a leaf labeled with this constant. Let $f = M_1 + M_2 + \cdots + M_t$. Consider any maximal monomial M_i of f. Let $M_i = x_{i_1} \cdots x_{i_{d'}}$ where $i_1 < i_2 < \cdots < i_{d'}$. The tree T_f has all the variables $x_{i_1}, \cdots, x_{i_{d'}}$ at the first d' levels of the tree. That is, the first d' levels of the tree is a complete tree where the label of all the nodes at level j is x_{i_j}. So every $\xi_1, \ldots, \xi_{d'} \in \{0,1\}$ leads to a different vertex at level d' in T_f from which we recursively attach the decision tree T_g where $g = f_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_{d'}} \leftarrow \xi_{d'}}$.

We now prove:

▶ **Lemma 4.** *Let f be a degree-d polynomial. Then, for $h = d(d-1)/2$:*

1. *If f is a depth-d decision tree, then T_f is depth h-decision tree.*
2. *If f is ϵ-far from every depth $h+1$ decision tree according to a distribution $\mathcal{D}$ then, for a random assignment a drawn according to the distribution $\mathcal{D}$, with probability at least ϵ, the path that a takes in T_f reaches depth $h+1$.*

Proof. **1.** follows immediately from Lemma 3.

To prove **2.**, let T'_f be the tree T_f where every vertex of depth $h+1$ is changed to a leaf labeled with 0. Since f is ϵ-far from every depth $h+1$ decision tree according to the distribution $\mathcal{D}$, it is ϵ far from T'_f. Since the leaves of T'_f of depth at most h correctly compute f, the probability that a random assignment a chosen according to distribution $\mathcal{D}$ ends up in a leaf of depth at most h is less than $1-\epsilon$. This completes the proof. ◀

3.2 The Tester

In this section, we prove Theorem 1. To that end, we start by the following Lemma:

▶ **Lemma 5.** *We have*

1. *There is an algorithm that for a degree-d polynomial f makes $q = \tilde{O}(2^{2d}) + 2^d \log n$ queries, runs in time $O(qn)$ and finds the relevant variables of f.*
2. *There is an algorithm that for a degree-d polynomial over 2^d variables X makes $q' = \tilde{O}(2^d)$ queries, runs in time $O(q'n)$ and finds a maximal monomial in f.*

The proof of Lemma 5 is given in subsection 3.3. This immediately gives the following result that we need for our tester.

▶ **Lemma 6.** *Let f be a sparse-3^d degree-d polynomial over 2^d variables. Let $h = d(d-1)/2$. Given the relevant variables of f, for any $a \in \{0,1\}^n$, the path that a takes in T_f up to depth at most $h+1$ can be computed in $\tilde{O}(2^d)h$ time.*

The tester's paradigm is as follows. First, the tester finds the relevant variables of f. If the number of relevant variables exceeds 2^d, then the tester rejects. The tester then, for $t = O(1/\epsilon)$ assignments $a^{(1)}, \ldots, a^{(t)}$ drawn according to the distribution $\mathcal{D}$, finds the route of each $a^{(i)}$ in T_f. If no maximal monomial of size at most d can be found then the tester rejects. If one of the routes exceeds depth $h = d(d-1)/2$, the algorithm rejects. Otherwise it accepts. Each route takes time $\tilde{O}(2^d)$. So the number of queries is $q = \tilde{O}(2^{2d})/\epsilon + 2^d \log n$ and the time is $O(qn)$. We now use the reduction of Bshouty in [6] to make the query complexity independent of n and get the result. See Lemma 26 in Appendix A.

3.3 Proof of Lemma 5

In this subsection, we prove Lemma 5. We show how to find the relevant variables and a maximal monomial of any degree-d polynomial.

The following is a very well known result [8]:

▶ **Lemma 7.** *For any non-constant degree-d polynomial f over F_2, we have $\boldsymbol{Pr}[f(x) \neq f(0)] \geq 1/2^d$.*

The following is a well known result in learning theory. We prove it for completeness.

▶ **Lemma 8.** *There is an algorithm that given any degree-d polynomial f over v variables and a set X of some of its relevant variables, asks $2^d \log(1/\delta) + \log v$ queries and, with probability at least $1-\delta$, decides if the variables in X are all its relevant variables, and if not, finds a new relevant variable of f.*

Proof. Let $X' = \{x_{i_1}, \ldots, x_{i_t}\}$ be the set of variables that are not in X. Define $g = f + f_{|X' \leftarrow 0}$. Since g is of degree at most d, by Lemma 7, with $2^d \log(1/\delta)$ queries to g, with probability at least $1-\delta$, we can decide if g is a constant function. If not, we get an assignment a such that $g(a) \neq g(0)$. If g is constant function $\tau \in \{0,1\}$, then $f(x) = f_{|X' \leftarrow 0} + \tau$ and f is independent of X'. So, the variables in X are all the relevant variables of f

If $g(a) \neq g(0)$, then since $g(0) = 0$, we have $f(a) \neq f_{|X' \leftarrow 0}(a) = f(a_{|X' \leftarrow 0})$. Now recursively flip half of the entries of a that differ from $a_{|X' \leftarrow 0}$ and ask a query and keep the two assignments that have different values in f. Eventually, we get an entry $a_{i_{k+1}}$ that flipping it changes the value of the function. Then, $x_{i_{k+1}}$ is relevant variable in f. Now $x_{i_{k+1}} \notin X$ is a new relevant variable because the entries of each $x_{i_j} \in X$ in a agree with the value of the same entry in $a_{|X' \leftarrow 0}$. The number of queries in this procedure is at most $\log v$. This completes the proof. ◀

Therefore, for degree-d polynomial, choosing confidence $\delta/2^d$ in the above algorithm, with probability at least $1 - \delta$, we find the first 2^d relevant variables of f using $\tilde{O}(2^{2d}) \log(1/\delta) + 2^d \log n$ queries. If f has more than 2^d relevant variables, then f is not a depth-d decision tree and the tester rejects.

We now prove

▶ **Lemma 9.** *Let f be a degree-d polynomial and X be the set of its relevant variables. Let $M = x_{i_1} \cdots x_{i_k}$ be a sub-monomial of some monomial of f (M is not necessarily a monomial of f). There is an algorithm that asks $O(2^d \log(1/\delta) + 2^k \log |X|)$ queries and, with probability at least $1 - \delta$, decides if M is a maximal monomial of f, and if it is not, it finds a new variable $x_{i_{k+1}}$ such that $M' = x_{i_1} \cdots x_{i_k} x_{i_{k+1}}$ is a sub-monomial of some monomial of f.*

Proof. Define the function $G(x) = 1 + \sum_{(\xi_1,\ldots,\xi_k) \in \{0,1\}^k} f_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x)$. We prove the following:

1. A query to G can be simulated by 2^k queries to f.
2. G is a polynomial of degree at most $d - k$.
3. M is maximal monomial of f if and only if $G = 0$.
4. If $G \neq 0$, then for any relevant variable $x_{i_{k+1}}$ of G, $M' = x_{i_1} \cdots x_{i_k} x_{i_{k+1}}$ is a sub-monomial of some monomial of f.

The first item is obvious. We prove 2-4. Since M is a sub-monomial of some monomial of f, we have $f = Mg + h$, where g is a polynomial of degree at most $d - k$ (independent of $x_{i_1}, \ldots, x_{i_k}$) and h is a polynomial of degree at most d that M is not sub-monomial of any of its monomials. Notice that M is maximal monomial of f if and only if $g = 1$. For a monomial M'' in h, we have that some variable in M, say w.l.o.g. x_{i_1}, is not in M'' and therefore,

$$\sum_{(\xi_1,\ldots,\xi_k) \in \{0,1\}^k} M''_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x) = \sum_{\xi_1 \in \{0,1\}} \sum_{(\xi_2,\ldots,\xi_k) \in \{0,1\}^{k-1}} M''_{|x_{i_2} \leftarrow \xi_2, \cdots, x_{i_k} \leftarrow \xi_k}(x) = 0.$$

Denote by $\xi := (\xi_1, \cdots, \xi_k) \in \{0,1\}^k$, then we can write:

$$\begin{aligned}
G(x) + 1 &= \sum_{\xi \in \{0,1\}^k} f_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x) \\
&= g(x) \sum_{\xi \in \{0,1\}^k} M_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x) + \sum_{\xi \in \{0,1\}^k} h_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x) \\
&= g(x) + \sum_{(M'' \text{ monomial in } h)} \sum_{(\xi_1,\ldots,\xi_k) \in \{0,1\}^k} M''_{|x_{i_1} \leftarrow \xi_1, \cdots, x_{i_k} \leftarrow \xi_k}(x) \\
&= g(x).
\end{aligned}$$

Hence, $f = MG + M + h$ and the results 2-4 follows.

By Lemma 8, there is an algorithm that asks $2^{d-k} \log(1/\delta) + \log |X|$ queries to G (and therefore, $O(2^d \log(1/\delta) + 2^k \log |X|)$ queries to f) and, with probability at least $1 - \delta$, either decides that $G = 0$, in which case M is maximal monomial, or finds a new relevant variable $x_{i_{k+1}}$ of G, in which case $M' = x_{i_1} \cdots x_{i_k} x_{i_{k+1}}$ is a sub-monomial of some monomial of f. ◀

For degree-d polynomials with $|X| \leq 2^d$ relevant variables, we choose confidence $\delta/2^d$ in the above algorithm and then, with probability at least $1-\delta$, we find a maximal monomial of f.

4 A Tester for Size-s Decision Tree

In this section we prove:

▶ **Theorem 10.** *There is a (uniform-distribution) tester that makes $q = poly(s, 1/\epsilon)$ queries to unknown function f, runs in $poly(s, 1/\epsilon)n$ time and*
1. *Accepts w.h.p if f is a size-s decision tree.*
2. *Rejects w.h.p if f is ϵ-far from size-$(s/\epsilon)^{O(\log(s/\epsilon))}$ decision trees.*

4.1 Preliminary Results

For a Boolean function f, the *constant-depth of f*, $cd(f)$, is the minimum number ℓ of variables $X = \{x_{j_1}, \cdots, x_{j_\ell}\}$ such that $f_{|X\leftarrow 0}$ is a constant function. We define $\mathcal{M}(f)$ the set of all monomials in the minimum size polynomial representation of f. Since minimum size polynomial representation of a Boolean function is unique, $\mathcal{M}(f)$ is well defined. For a set of monomials S, we denote $\Sigma S = \sum_{M\in S} M$. Notice that $\Sigma\mathcal{M}(f) = f$ and $\mathcal{M}(\Sigma S) = S$. For any Boolean function and an interval I (such as $[d] = \{1, 2, \ldots, d\}$, $[d_1, d_2] = [d_2]\backslash[d_1 - 1]$ or $(d_1, d_2] = [d_2]\backslash[d_1]$), we define $f^I = \sum_{M\in\mathcal{M}(f) \text{ and } |M|\in I} M$ where $|M|$ is the number of variables in M. We first prove:

▶ **Lemma 11.** *For any $S' \subseteq \mathcal{M}(f)$, we have $cd(\Sigma S') \leq cd(f)$. In particular, for any interval I, we have $cd(f^I) \leq cd(f)$.*

Proof. Let $cd(f) = r$. Then, there is a set $X = \{x_{j_1}, \cdots, x_{j_r}\}$ such that $f_{|X\leftarrow 0}$ is constant. Therefore, for every non-constant $M \in \mathcal{M}(f)$, we have $M_{|X\leftarrow 0} = 0$. Then, $(\Sigma S')_{|X\leftarrow 0}$ is constant and $cd(\Sigma S') \leq cd(f)$. ◀

▶ **Lemma 12.** *Let f be any Boolean function. If $cd(f) \leq \ell$, then there is a variable x_i that appears in at least $1/\ell$ fraction of the non-constant monomials of f.*

Proof. If $cd(f) \leq \ell$, then there is a set $X = \{x_{j_1}, \cdots, x_{j_\ell}\}$ such that $f_{|X\leftarrow 0}$ is a constant function. This implies that for every non-constant monomial, there is $x_i \in X$ that appears in it. By the pigeonhole principle the result follows. ◀

Let $a \in \{0,1\}^n$ be a random uniform assignment. Consider, $T(x) = f(x + a)$. Then:

▶ **Lemma 13.** *Let f be a size-s decision tree and let $T(x) = f(x + a)$ for a random uniform assignment $a \in \{0,1\}^n$. For a random uniform $\xi_1, \ldots, \xi_j \in \{0,1\}$ and any variables $x_{i_1}, \ldots, x_{i_j}$ where each i_ℓ may depend on T, a, $i_1, \ldots, i_{\ell-1}$ and $\xi_1, \ldots, \xi_{\ell-1}$ but is independent of $\xi_\ell, \ldots, \xi_j$ and $q = (x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j)$, with probability at least $1 - s2^{-h}$, $cd\left(T_{|q}\right) \leq h$.*

Proof. We have $T_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_j}\leftarrow\xi_j}(0) = T(0_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_j}\leftarrow\xi_j}) = f(a + 0_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_j}\leftarrow\xi_j})$. Since $b := a + 0_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_j}\leftarrow\xi_j}$ is random uniform in $\{0,1\}^n$, the path that b takes in the computation of $f(b)$ is a random uniform path in f. With probability at least $1 - s2^{-h}$, this path reaches a leaf at depth less than or equal to h in f. Therefore, with probability at least $1 - s2^{-h}$, there are $h' \leq h$ variables $x_{j_1}, \ldots, x_{j_{h'}}$ (the variables in this path) such that $f_{x_{j_1}\leftarrow b_{j_1},\ldots,x_{j_{h'}}\leftarrow b_{j_{h'}}}$ is constant, say $\tau \in \{0,1\}$. Let $J = \{j_1, j_2, \ldots, j_{h'}\}$ and $I = \{i_1, i_2, \ldots, i_j\}$ and suppose, w.l.o.g, $J \cap I = \{i_1, \ldots, i_r\}$. Then,

$$\tau = f_{|x_{j_1}\leftarrow b_{j_1},\ldots,x_{j_{h'}}\leftarrow b_{j_{h'}}}(x) = f_{|x_{j_1}\leftarrow b_{j_1},\ldots,x_{j_{h'}}\leftarrow b_{j_{h'}}}(x+a)$$
$$= T_{|x_{j_1}\leftarrow b_{j_1}+a_{j_1},\ldots,x_{j_{h'}}\leftarrow b_{j_{h'}}+a_{j_{h'}}} = \left(T_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_r}\leftarrow\xi_r}\right)_{|(J\setminus I)\leftarrow 0}$$

and thus,

$$(T_{|q})_{|(J\setminus I)\leftarrow 0} = (T_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_j}\leftarrow\xi_j})_{|(J\setminus I)\leftarrow 0}$$
$$= (T_{|x_{i_1}\leftarrow\xi_1,\ldots,x_{i_r}\leftarrow\xi_r})_{|(J\setminus I)\leftarrow 0, x_{i_{r+1}}\leftarrow\xi_{r+1},\ldots,x_{i_j}\leftarrow\xi_j} = \tau.$$

Therefore, $cd\left(T_{|q}\right) \le |J\setminus I| \le h' \le h$. ◀

Let

$$r = \log(s/\epsilon). \tag{1}$$

The tester uses Lemma 13, $poly(\log(s/\epsilon))/\epsilon$ times, therefore we can choose $h = 2r$ which, by union bound, adds a failure probability of $(poly(\log(s/\epsilon))/\epsilon)\cdot s2^{-h} = \tilde{O}(\epsilon/s)$ to the tester. Let E_1 be the event

$$E_1: \quad (\forall q)\ cd(T_{|q}) \le 2r, \tag{2}$$

for all the q that are generated in the tester.

For the rest of this section, we let f be a size-s decision tree. Let $f = f_1 + f_2 + \cdots + f_s$ be the disjoint-terms sum representation of f.[1] Let $T_i = f_i(x+a)$ for $i \in [s]$. It is easy to see that $T = T_1 + T_2 + \cdots + T_s$ is disjoint-terms sum representation of T. We denote by T_i^+ the conjunction of the non-negated variables in T_i. We prove:

▶ **Lemma 14.** *Let λ be any constant. For a random uniform a, with probability at least $1 - s(\epsilon/s)^\lambda$ the following event $E_2(\lambda)$ holds*

$$E_2(\lambda): \quad \textit{For every } i, \textit{ if } |T_i| > 16\lambda r \textit{ then } |T_i^+| \ge 4\lambda r. \tag{3}$$

Proof. Since $T_i(x) = f_i(x+a)$ and a is random uniform, each variable in T_i is positive with probability 1/2. By Chernoff bound the result follows. ◀

To change the disjoint-terms representation of T to polynomial representation, we take every term T_i and expand it to sum of monomials[2]. A monomial that is generated from even number of different terms will not appear in the polynomial, while, those that are generated from odd number of different terms will appear in the polynomial.

Let $M_1 + M_2 + \cdots + M_\ell$ be the multivariate polynomial representation of T, where $|M_1| \le |M_2| \le \cdots \le |M_\ell|$. Note that ℓ can be exponential in n. We say that M_i *is generated by* T_j if j is the smallest integer for which T_j generates M_i. The following is a trivial result:

▶ **Lemma 15.** *If M_i is generated by T_j, then $|T_j| \ge |M_i| \ge |T_j^+|$ and T_j^+ is a sub-monomial of M_i. That is, $M_i = T_j^+ M_i'$ for some monomial M_i'.*

---

[1] Every size-s decision tree can be represented as a sum of terms $T_1 + T_2 + \cdots + T_{s'}$, $s' \le s$, where $T_i \wedge T_j = 0$ for every $i \ne j$. The number of terms s' is the number of leaves labeled with 1. See for example [8]. Here we assume $s' = s$ because we can always change a term t to $tx_j + t\overline{x_j}$.

[2] For example $x_1x_2\bar{x}_3\bar{x}_4 = x_1x_2(x_3+1)(x_4+1) = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2x_4 + x_1x_2$.

▶ **Lemma 16.** *If $E_2(\lambda)$ in (3) holds, then the number of monomials M_i of size at most $4\lambda r$ is at most $s(s/\epsilon)^{16\lambda}$.*

Proof. By Lemma 14, the monomials that has size at most $4\lambda r$ are generated from terms of size at most $16\lambda r$. We have at most s terms of size at most $16\lambda r$ and each one generates at most $2^{16\lambda r}$ monomials. So we have at most $s(s/\epsilon)^{16\lambda}$ such monomials. ◀

▶ **Lemma 17.** *If $E_2(\lambda)$ in (3) holds, then there are s monomials $N_1, N_2, \ldots, N_s$, each of size at least $4\lambda r$, such that for every monomial M_i (of T) of size at least $16\lambda r$, there is N_j where $M_i = N_j M_i'$ for some monomial M_i'.*

Proof. Let $N_i = T_i^+$ if $|T_i^+| \geq 4\lambda r$n and $N_i = x_1 x_2 \cdots x_n$ otherwise. Let M_i be a monomial of T of size at least $16\lambda r$. By Lemma 15, M_i is generated by a monomial T_j of size at least $16\lambda r$. By Lemma 14, $|T_j^+| \geq 4\lambda r$. By Lemma 15, $N_j = T_j^+$ is a sub-monomial of M_i. ◀

We remind the reader that for an interval R, T^R is the sum of the monomials M of T of size $|M| \in R$. For a set of monomials A, we denote $\vee A = \vee_{M \in A} M$. We also need the following lemma:

▶ **Lemma 18.** *Suppose $E_2(\lambda)$ holds. Let $P = \vee \mathcal{M}(T^{(16\lambda r, s]})$. For a random uniform $\xi_1, \ldots, \xi_j \in \{0,1\}$ and any variables $x_{i_1}, \ldots, x_{i_j}$ where each i_ℓ may depend on T, $i_1, \ldots, i_{\ell-1}$ and $\xi_1, \ldots, \xi_{\ell-1}$ but independent of $\xi_\ell, \ldots, \xi_j$ and $q = (x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j)$, with probability at least $1 - s(\epsilon/s)^{2\lambda}$, $\boldsymbol{Pr}\left[P_{|q} = 1\right] \leq s\left(\frac{\epsilon}{s}\right)^{2\lambda}$.*

Proof. By Lemma 17, we have $P = N_1 P_1 \vee N_2 P_2 \vee \cdots \vee N_s P_s$, where P_i is a Boolean function and N_i is a monomial of size at least $4\lambda r$ for each $i \in \{1, \ldots, s\}$. The probability that each $(N_i)_{|q}$ is not zero and is of size at most $2\lambda r$ is at most $2^{-2\lambda r} = (\epsilon/s)^{2\lambda}$. Since, for all $i \in [s]$, $(N_i)_{|q}$ is zero or $|(N_i)_{|q}| > 2\lambda r$ implies $\mathbf{Pr}[P_{|q} = 1] \leq s(\epsilon/s)^{2\lambda}$, the result follows. ◀

4.2 The Tester

In this subsection, we give the tester and prove its correctness. Recall that $r = \log(s/\epsilon)$. Let $c \geq 2$ be any constant. The tester first chooses a random uniform $a \in \{0,1\}^n$ and defines $T(x) = f(x+a)$. Then, it learns all the monomials of size $16r'$ where[3] $r' = 16cr$. This can be done by the algorithm in [8] in $poly(n, s/\epsilon)$ time and queries. We show later how to eliminate n in the query complexity. Then, the tester splits the monomials of size at most $16r'$ to monomials of size less or equal to r' (the function F) and those that have size between r' and $16r'$ (the function G). The tester performs $O(1/\epsilon)$ random walks in a decision tree. For each stage j, the set H_j is defined in a way that (1) it is a subset of the monomials of the function $F_{|x_{i_1} \leftarrow \xi_1, \ldots, x_{i_{j-1}} \leftarrow \xi_{j-1}}$ and (2) it contains a variable that appears in at least $1/(2r)$ fraction of the monomials. At each stage j, the tester deterministically chooses a variable x_{i_j} with the smallest index i_j that appears in at least $1/(2r)$ fraction of the monomials of ΣH_j and chooses a random $\xi_j \in \{0,1\}$ for x_{i_j}. See the details in Algorithm 1.

Although this decision tree is not the decision tree of F, we can still show that when f is size-s decision tree, with probability at least $2/3$ the random walk ends after $O(\log^2(s/\epsilon))$ steps and then the tester accepts. When it is ϵ-far from any size-$(s/\epsilon)^{O(\log(s/\epsilon))}$ decision tree then, with probability at least $2/3$, something goes wrong (the learning algorithm fails or no variable appears in at least $1/(2r)$ fraction of the monomials of H_j) or the random walk does not end after $\Omega(\log^2(s/\epsilon))$ steps and then it rejects.

[3] Here c can be 2. We kept it to show the effect of this constant on the success probability of the tester.

Algorithm 1 : Test(f) - A tester for size-s decision tree.

Input: Black box access to f
Output: Accept or Reject

$T \leftarrow f(x+a)$ for random uniform $a \in \{0,1\}^n$.
Learn $T^{[16r']}$. If FAIL then Reject.
$F \leftarrow T^{[r']}$; $G \leftarrow T^{(r',16r']}$; $H_0 \leftarrow \mathcal{M}(F)$; $L_0 \leftarrow \mathcal{M}(G)$.
for $i = 1$ to $40/\epsilon$ **do**
    $j \leftarrow 0$; $q_0 \leftarrow$ Empty sequence.
    while ΣH_j is not constant and $j < 2^{10}c(\log^2(s/\epsilon))$ **do**
        $j \leftarrow j+1$.
        Find a variable x_{i_j} with the smallest index that appears in at least $1/(2r)$ fractions of the monomials in H_{j-1}.
        If no such variable exists then Reject.
        Choose a random uniform $\xi_j \in \{0,1\}$.
        $q_j \leftarrow (q_{j-1}; x_{i_j} \leftarrow \xi_j)$.
            ▷ I.e., add to the list of substitutions q_{j-1} the substitution $x_{i_j} \leftarrow \xi_j$.
        $H_j \leftarrow \mathcal{M}((\Sigma H_{j-1})_{|x_{i_j}\leftarrow\xi_j}) \backslash \mathcal{M}((\Sigma L_{j-1})_{|x_{i_j}\leftarrow\xi_j})$
        $L_j \leftarrow \mathcal{M}((\Sigma L_{j-1})_{|x_{i_j}\leftarrow\xi_j}) \backslash \mathcal{M}((\Sigma H_{j-1})_{|x_{i_j}\leftarrow\xi_j})$
    end while
    if $j = 2^{10}c(\log^2(s/\epsilon))$ **then**
        Reject.
    end if
end for
if $\mathbf{Pr}[T_{|q_j} = 1]$ is in $[\epsilon/4, 1-\epsilon/4]$ **then**
    Reject
end if
Accept.

The tester query complexity is $poly(n, s/\epsilon)$. We use the reduction from [6] to change the query complexity to $poly(s/\epsilon)$.

▶ **Lemma 19.** *Assume that the event $E_2(16c)$ in (3) holds. Let $F = T^{[r']}$ and $G = T^{(r',16r']}$. Let $x_{i_1}, \ldots, x_{i_j}$ be variables, $\xi_1, \ldots, \xi_j$ be random uniform values in $\{0,1\}$, $q_j = (x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j)$, $q_{j+1} = (q_j, x_{i_{j+1}} \leftarrow \xi_{j+1})$, H_j, H_{j+1} and L_j as defined in the procedure **Test**(f) in Algorithm 1. Then, with probability at least $1 - s(\epsilon/s)^{3c}$, we have*

1.

$$H_j \subseteq \mathcal{M}\left((T_{|q_j})^{[r']}\right). \tag{4}$$

Let E be the event that (4) holds for all $j \le 2^{10}c\log^2(s/\epsilon)$ and all the $40/\epsilon$ random walks of the tester. Then, $\mathbf{Pr}[E] \ge 1 - (\epsilon/s)^{2c}$.

Assuming that E holds, then,

2. *There is a variable $x_{i_{j+1}}$ that appears in $1/(2r)$ fraction of the monomials in H_j.*
3. *If $\xi_{j+1} = 0$, then $|H_{j+1}| \le (1 - 1/(2r))|H_j|$.*
4. *If $\xi_{j+1} = 1$, then $|H_{j+1}| \le |H_j|$.*

Proof. Let $T = F + G + W$ such that $W = T^{(16r',s]}$. We first show that with probability at least $1 - s(\epsilon/s)^{3c}$, we have $(W_{|q_j})^{[r']} = 0$. Consider $N_1, N_2, \ldots, N_s$ in Lemma 17. Every monomial in W is of the form MN_i for some monomial M and $i \in [s]$. We also have

$|N_i| \geq 4r'$ for all $i \in [s]$. The probability that for some $i \in [s]$ we have that $(N_i)_{|q_j}$ is not zero and of size at most r' is at most $s2^{-3r'} \leq s(\epsilon/s)^{3c}$. Therefore, with probability at least $1 - s(\epsilon/s)^{3c}$, we have $(W_{|q_j})^{[r']} = 0$.

By induction, we prove that:

1. $H_j \cap L_j = \varnothing$.
2. H_j contains monomials of size at most r' and
3.

$$\Sigma H_j + \Sigma L_j = F_{|q_j} + G_{|q_j}. \tag{5}$$

For $j = 0$ the result follows from the fact that $H_0 = \mathcal{M}(T^{[r']}) = \mathcal{M}(F)$, $L_0 = \mathcal{M}(T^{(r',16r']}) = \mathcal{M}(G)$ and $q_0 = ()$ is the empty sequence. Assume the above hold for $j-1$. We now prove it for j. 1 and 2 follow immediately from steps (13) and (14) in the algorithm. For 3, we have[4]

$$\begin{aligned} H_j + L_j = H_j \cup L_j &= \mathcal{M}((\Sigma H_{j-1})_{|x_{i_j} \leftarrow \xi_j}) + \mathcal{M}((\Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j}) \\ &= \mathcal{M}((\Sigma H_{j-1})_{|x_{i_j} \leftarrow \xi_j} + (\Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j}) = \mathcal{M}((\Sigma H_{j-1} + \Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j}) \\ &= \mathcal{M}((F_{|q_{j-1}} + G_{|q_{j-1}})_{|x_{i_j} \leftarrow \xi_j}) = \mathcal{M}(F_{|q_j} + G_{|q_j}), \end{aligned}$$

which implies the result. Since H_j contains the monomials of size at most r', $(F_{|q_j})^{[r']} = F_{|q_j}$ and $H_j \cap L_j = \varnothing$, we get $H_j \subseteq (H_j + L_j)^{[r']} = S\left(F_{|q_j} + (G_{|q_j})^{[r']}\right)$. Then, with probability at least $1 - s(\epsilon/s)^{3c}$, we have

$$\begin{aligned} (T_{|q_j})^{[r']} &= (F_{|q_j})^{[r']} + (G_{|q_j})^{[r']} + (W_{|q_j})^{[r']} = F_{|q_j} + (G_{|q_j})^{[r']} + (W_{|q_j})^{[r']} \\ &= F_{|q_j} + (G_{|q_j})^{[r']}, \end{aligned}$$

and then, $H_j \subseteq S\left(F_{|q_j} + (G_{|q_j})^{[r']}\right) = S\left((T_{|q_j})^{[r']}\right)$. This completes the proof of 1.

By Lemma 11, Equation (2) and case 1 of this Lemma, we have: $cd(\Sigma H_j) \leq cd(T^{[r']}_{|q_j}) \leq cd(T_{|q_j}) \leq 2r$. Therefore, by Lemma 12 the result 2 follows.

We now prove (3-4). Since $H_{j+1} = \mathcal{M}((\Sigma H_j)_{|x_{i_{j+1}} \leftarrow \xi_{j+1}}) \backslash \mathcal{M}((\Sigma L_j)_{|x_{i_{j+1}} \leftarrow \xi_{j+1}}) \subseteq \mathcal{M}((\Sigma H_j)_{|x_{i_{j+1}} \leftarrow \xi_{j+1}})$, we get 4. Since $x_{i_{j+1}}$ appears in more than $1/(2r)$ fraction of the monomials in H_j, we get 3. ◀

Before we prove the next result, we give some more notations. For a set of monomials A and $q = (x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j)$, we denote $A_{|q} = \{M_q | M \in A\}$. Recall that $\vee A = \vee_{M \in A} M$. The following properties are easy to prove: Let g be a Boolean function, A, B sets of monomials, $q = (x_{i_1} \leftarrow \xi_1, \ldots, x_{i_j} \leftarrow \xi_j)$ and $q' = (x_{i'_1} \leftarrow \xi'_1, \ldots, x_{i'_j} \leftarrow \xi'_j)$. Then:(I) $(A_{|q})_{|q'} = A_{|q,q'}$, (II) $\mathcal{M}(g_{|q}) \subseteq \mathcal{M}(g)_{|q}$, (III) $(\vee A)_{|q} = \vee A_{|q}$, (IV) if $A \subseteq B$ then[5] $\vee A \Rightarrow \vee B$ and (V) $\Sigma A \Rightarrow \vee A$ and $g \Rightarrow \vee \mathcal{M}(g)$.

Using the above notations and results we prove:

▶ **Lemma 20.** *Suppose events $E_2(c)$ and $E_2(16c)$ in (3) hold. If $\Sigma H_j = \eta$ is a constant function, $\eta \in \{0, 1\}$, then with probability at least $1 - (\epsilon/s)^{2c-1}$, $\boldsymbol{Pr}[T_{|q_j} \neq \eta] \leq s\left(\frac{\epsilon}{s}\right)^{2c-1}$.*

[4] The operation + for sets is the symmetric difference of sets.

[5] $f \Rightarrow g$ means if $f(x) = 1$ then $g(x) = 1$. In particular, $\mathbf{Pr}[f = 1] \leq \mathbf{Pr}[g = 1]$.

Proof. Let $T = F + G + W$, where $F = T^{[r']}$, $G = T^{(r',16r']}$ and $W = T^{(16r',s]}$. We have,

$$\begin{aligned}\mathbf{Pr}[T_{|q_j} \neq \eta] = \ & \mathbf{Pr}[F_{|q_j} + G_{|q_j} + W_{|q_j} \neq \eta] = \ \mathbf{Pr}[\Sigma H_j + \Sigma L_j + W_{|q_j} \neq \eta] \qquad \text{By (5)} \\ = \ & \mathbf{Pr}[\Sigma L_j + W_{|q_j} \neq 0] \leq \ \mathbf{Pr}[\Sigma L_j = 1] + \ \mathbf{Pr}[W_{|q_j} = 1].\end{aligned}$$

Since $W = T^{(16r',s]} \Rightarrow \vee\mathcal{M}(T^{(16r',s]})$, by Lemma 18, we have, with probability at least $1 - s(\epsilon/s)^{32c}$, $\mathbf{Pr}[W_{|q_j} = 1] \leq s\left(\frac{\epsilon}{s}\right)^{32c}$. Since

$$\begin{aligned}L_j &= \mathcal{M}((\Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j}) \backslash \mathcal{M}((\Sigma H_{j-1})_{|x_{i_j} \leftarrow \xi_j}) \subseteq \mathcal{M}((\Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j}) \\ &\subseteq \mathcal{M}(\Sigma L_{j-1})_{|x_{i_j} \leftarrow \xi_j} = (L_{j-1})_{|x_{i_j} \leftarrow \xi_j},\end{aligned}$$

we can conclude that, $L_j \subseteq (L_0)_{|q} = \mathcal{M}(G)_{|q} \subseteq \mathcal{M}(T^{(r',s]})_{|q} = \mathcal{M}(T^{(16cr,s]})_{|q}$, which implies that $\Sigma L_j \Rightarrow \vee L_j \Rightarrow \vee\mathcal{M}(T^{(16cr,s]})_{|q} = (\vee\mathcal{M}(T^{(16cr,s]}))_{|q}$. Therefore, by Lemma 18, with probability at least $1 - s(\epsilon/s)^{2c}$, $\mathbf{Pr}[\Sigma L_j = 1] \leq \ \mathbf{Pr}[(\vee\mathcal{M}(T^{(16cr,s]}))_{|q}] \leq s\left(\frac{\epsilon}{s}\right)^{2c}$. ◀

▶ **Lemma 21.** *Suppose the events $E_2(4c)$, $E_2(16c)$ and E hold. If f is a size-s decision tree then, with probability at least $1 - (\epsilon/s)^{O(s/\epsilon)}$, ΣH_k is constant for $k \leq 2^{10}\log^2(s/\epsilon)$.*

Proof. By Lemma 16, we have $|H_0| = |\mathcal{M}(F)| = |\mathcal{M}(T^{[r']})| \leq s(s/\epsilon)^{64c}$. By Lemma 19, we have that with probability $1/2$, $\xi_j = 1$ and then $|H_{j+1}| \leq |H_j|$. And, with probability $1/2$, $\xi_j = 0$ and then $|H_{j+1}| \leq (1 - 1/(2r))|H_j|$. Therefore, when $\xi_1, \ldots, \xi_t$ contains $2r\ln(s(s/\epsilon)^{64c}) \leq 2^8 c\log^2(s/\epsilon)$ zeros, then ΣH_k will be constant for $k \leq t$. The probability of $\xi_i = 0$ is $1/2$, and thus, by Chernoff bound the result follows. ◀

▶ **Lemma 22.** *If f is a size-s decision tree, then with probability at least $1 - poly(\epsilon/s)$, the tester accepts.*

Proof. By Lemma 21, with probability at least $1 - poly(\epsilon/s)$, the tester does not reject inside the Repeat loop. By Lemma 20, with probability at least $1 - poly(\epsilon/s)$, $\mathbf{Pr}[T_{|q_j} \neq \eta] \leq poly(\epsilon/s)$, hence, with probability at least $1 - poly(\epsilon/s)$, the tester will not reject in line 20. ◀

▶ **Lemma 23.** *Let $R = 2^{10}c\log^2(s/\epsilon)$. If f is ϵ-far from every size-2^R decision tree, then with probability at least $2/3$, the tester rejects.*

Proof. If f is ϵ-far from every size-2^R decision tree, then $T = f(x + a)$ is ϵ-far from every size-2^R decision tree.

Consider the tree T^* that is generated in the tester for all possible random walks, where each node in the tree is labeled with the variable x_{i_j} that appears in at least $1/(2r)$ of the monomials in H_{j-1} if such variable exists, and is labeled with Reject when the algorithm reaches Reject. In the tree, we will have three types of nodes that are labeled with Reject. Type I are nodes where there is no variable that appears in at least $1/(2r)$ fractions of the monomials of H_{j-1}. Type II are the nodes that are of depth $R + 1$, and Type III are the nodes of depth less than R where ΣH_j is constant η and $\mathbf{Pr}[T_{|q_j} = \eta] \geq \epsilon/4$.

If, with probability at least $\epsilon/4$, a random walk in the tree T^* reaches a Reject node, then the probability that the tester rejects is $1 - \left(1 - \frac{\epsilon}{4}\right)^{40/\epsilon} \geq \frac{2}{3}$, and we are done.

Suppose, for the contrary, this is not true. Then, define a decision tree T' that is equal to T^*, where each Reject node is replaced with a leaf labelled with 0, and each other other leaf is labeled with 0 if $\mathbf{Pr}[T_{|q_j}] \leq \epsilon/4$ and 1 if $\mathbf{Pr}[T_{|q_j}] \geq 1 - \epsilon/4$. Then, T' is a depth-R tree (and therefore size-2^R tree). The probability that $T'(x)$ is not equal to $T(x)$ is less than the

probability that a random walk arrives to a Reject leaf, or if it arrives to a non-Reject leaf, then $T_{|q_j}(x)$ is not equal to the label in the leaf. Therefore, $\mathbf{Pr}[T'(x) \neq T(x)] \leq \frac{\epsilon}{4} + \frac{\epsilon}{4} < \epsilon$, a contradiction. ◀

As we said earlier, the above tester runs in $poly(n, 1/\epsilon)$ time and queries. We now use the reduction of Bshouty in [6] and get a tester that runs in time $poly(s, 1/\epsilon)n$ and makes $poly(s, 1/\epsilon)$ queries. See Lemma 26 in Appendix A.

References

1. Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 210–220, 2011. doi:10.1109/CCC.2011.31.
2. Guy Blanc, Jane Lange, Mingda Qiao, and Li-Yang Tan. Properly learning decision trees in almost polynomial time. *CoRR*, abs/2109.00637, 2021. arXiv:2109.00637.
3. Guy Blanc, Jane Lange, and Li-Yang Tan. Testing and reconstruction via decision trees. *CoRR*, abs/2012.08735, 2020. arXiv:2012.08735.
4. Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993. doi:10.1016/0022-0000(93)90044-W.
5. Nader H. Bshouty. Almost optimal testers for concise representations. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:156, 2019. URL: https://eccc.weizmann.ac.il/report/2019/156.
6. Nader H. Bshouty. Almost optimal testers for concise representations. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 5:1–5:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.5.
7. Nader H. Bshouty and Catherine A. Haddad-Zaknoon. Adaptive exact learning of decision trees from membership queries. In Aurélien Garivier and Satyen Kale, editors, *Algorithmic Learning Theory, ALT 2019, 22-24 March 2019, Chicago, Illinois, USA*, volume 98 of *Proceedings of Machine Learning Research*, pages 207–234. PMLR, 2019. URL: http://proceedings.mlr.press/v98/bshouty19a.html.
8. Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM J. Comput.*, 31(6):1909–1925, 2002. doi:10.1137/S009753979732058X.
9. Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 545–556, 2011. doi:10.1007/978-3-642-22006-7_46.
10. Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 549–558, 2007. doi:10.1109/FOCS.2007.32.
11. Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.
12. Michael J. Kearns and Dana Ron. Testing problems with sublearning sample complexity. *J. Comput. Syst. Sci.*, 61(3):428–456, 2000. doi:10.1006/jcss.1999.1656.
13. Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

A Reductions in Testing

The following reductions are deduced immediately from [5, 6].

We say that a class C is *closed under zero-one projection* if for every $f \in C$, every $i \in [n]$ and every $\xi \in \{0,1\}$ we have $f_{|x_i \leftarrow \xi} \in C$. We say that C is *symmetric* if for every permutation $\pi : [n] \to [n]$ and every $f \in C$ we have $f_\pi \in C$ where $f_\pi(x) := f(x_{\pi(1)}, \cdots, x_{\pi(n)})$. All the classes in this paper are closed under zero-one projection and symmetric.

The following results are proved in [6] (Theorem 2) when $H = C$. The same proof works for any $C \subseteq H$.

▶ **Lemma 24** ([6]). *Let $C \subseteq H$ be classes of Boolean functions (over n variables) that are symmetric sub-classes of k-$\mathbb{JUNTA}$ and are closed under zero-one projection. Suppose C is distribution-free learnable from H in time $T(n,\epsilon,\delta)$ using $Q(n,\epsilon,\delta)$ random example queries and $M(n,\epsilon,\delta)$ black-box queries. Then, there is a distribution-free two-sided adaptive algorithm for ϵ-testing C by H that runs in time $T(k,\epsilon/12,1/24) + O(mn)$ and makes $m = \tilde{O}\left(M(k,\epsilon/12,1/24) + k \cdot Q(k,\epsilon/12,1/24) + \frac{k}{\epsilon}\right)$ queries.*

▶ **Lemma 25** ([6]). *Let $C \subseteq H$ be classes of Boolean functions (over n variables) that are symmetric sub-classes of k-$\mathbb{JUNTA}$ and are closed under zero-one projection. Suppose C is learnable from H under the uniform distribution in time $T(n,\epsilon,\delta)$ using $Q(n,\epsilon,\delta)$ random example queries and $M(n,\epsilon,\delta)$ black-box queries. Then, there is a (uniform distribution) two-sided adaptive algorithm for ϵ-testing C by H runs in time $T(k,\epsilon/12,1/24) + O(mn)$ and that makes $m = \tilde{O}\left(M(k,\epsilon/12,1/24) + Q(k,\epsilon/12,1/24) + \frac{k}{\epsilon}\right)$ queries.*

The following is proved in [6] when $H = C$. The same proof gives the following result:

▶ **Lemma 26** ([6]). *Let C and $C \subseteq H$ be classes of Boolean functions that are symmetric sub-classes of k-$\mathbb{JUNTA}$ and are closed under zero-one projection. Suppose there is a tester $\mathcal{T}$ for $C_k = \{f \in C | f$ is independent on $x_{k+1}, \ldots, x_n\}$ such that*

1. *$\mathcal{T}$ is a two-sided adaptive ϵ-tester (resp. distribution-free ϵ-tester) that runs in time $T(k,\epsilon,\delta)$.*
2. *If $f \in C_k$ then, with probability at least $1-\delta$, $\mathcal{T}$ accepts.*
3. *If f is ϵ-far from every function in H_k (resp., with respect to $\mathcal{D}$) then, with probability at least $1-\delta$, $\mathcal{T}$ rejects.*
4. *$\mathcal{T}$ makes $Q(k,\epsilon,\delta)$ random example queries and $M(k,\epsilon,\delta)$ black-box queries.*

Then, there is a two-sided adaptive algorithm for ϵ-testing (resp., distribution-free algorithm for ϵ-testing) C by H that makes $m = \tilde{O}\left(M(k,\epsilon/12,1/24) + Q(k,\epsilon/12,1/24) + \frac{k}{\epsilon}\right)$ (resp., makes $m = \tilde{O}\left(M(k,\epsilon/12,1/24) + k \cdot Q(k,\epsilon/12,1/24) + \frac{k}{\epsilon}\right)$) queries and runs in time $T(k,\epsilon/12,1/24) + O(mn)$.