Proceedings of Machine Learning Research vol 98:1–28, 2019 30th International Conference on Algorithmic Learning Theory

Adaptive Exact Learning of Decision Trees from Membership Queries

Nader H. Bshouty BSHOUTY@CS.TECHNION.AC.IL
Department of Computer Science
Technion
Haifa, 32000, Israel

Catherine A. Haddad-Zaknoon CATHERINE@CAMPUS.TECHNION.AC.IL
Department of Computer Science
Technion
Haifa, 32000, Israel

Editors: Aurélien Garivier and Satyen Kale

Abstract

In this paper we study the adaptive learnability of decision trees of depth at most d from membership queries. This has many applications in automated scientific discovery such as drugs development and software update problem.

Feldman solves the problem in a randomized polynomial time algorithm that asks $\tilde{O}(2^{2d})\log n$ queries. Kushilevitz-Mansour solve it in a deterministic polynomial time algorithm that asks $2^{18d+o(d)}\log n$ queries. We improve the query complexity of both algorithms. We give a randomized polynomial time algorithm that asks $\tilde{O}(2^{2d})+2^d\log n$ queries and a deterministic polynomial time algorithm that asks $2^{5.83d}+2^{2d+o(d)}\log n$ queries.

Keywords: Decision Trees, Membership Queries, Adaptive Learning, Exact Learning

1. Introduction

Learning from membership queries, (Angluin, 1987), has fueled the interest of many papers due to its diverse applications in various areas including group testing (Du and Hwang, 1993), pooling design for DNA sequencing (Du and Hwang, 2006), blood testing (Dorfman, 1943), functional genomics (molecular biology) (King et al., 2004), chemical reactions (Angluin and Chen, 2008; Abasi et al., 2014), multi-access channel communications (Du and Hwang, 1993; Biglieri and Gyrfi, 2007), whole-genome shotgun sequencing (Alon et al., 2002), DNA physical mapping (Grebinski and Kucherov, 1998), game theory (Pelc, 2002), program synthesis (Bshouty et al., 2017), channel leak testing, codes, VLSI testing and AIDS screening and many others as in (Bshouty and Costa, 2016) and the references therein. Generally, in domains where queries can be answered by a non-human annotator, e.g. from experiments such as laboratory tests, learning from membership query may be a promising direction for automated scientific discovery.

In this paper, we consider the problem of learning Boolean decision trees from membership queries provided that the tree is of depth at most d. Learning decision trees from membership queries is equivalent to solving problems in widespread areas including drug development, software update problem, bioinformatics, chemoinformatics and many others.

© 2019 N.H. Bshouty & C.A. Haddad-Zaknoon.

1.1. Applications

In the sequel we bring two examples where learning decision tree in the exact learning model might be applied to improve scientific and technological research.

Drug development. The process of drugs discovery starts by searching for lead compounds in a huge library of chemical molecules (Grave et al., 2008; Dudek et al., 2006; Landrum et al., 2003). A compound is called *active* with respect to some (biological) problem if it exhibits a very high activity in a biological test or *assay*. For example, checking if some molecule binds with a specific protein or cancerous cell. The essential goal is to find, at a very early stage, some lead compounds that exhibit a very high activity in the assay. One of the famous methods to cope with this problem is building *Quantitive Structure-Activity Relationship* model or *QSAR*. The *QSAR* methodology focuses on finding a model, which correlates between *activity* and the compound structure. A molecule is described using a predefined set of features or *descriptors* selected according to chemical considerations. Decision trees are one of the standard ways used in drug research to model such relations. A membership query can be simulated by building a compound according to the specific descriptors defined by the algorithm's query and screening it against the biological assay. Compounds that exhibit high activity against the assay are related to as positive queries, while inactive compounds are related as negative ones.

The software update problem. Complicated software systems are made up of many interacting packages that must be deployed and coexist in the same context. One of the tough challenges system administrators face is to know which software upgrades are safe to apply to their particular environment without breaking the running and working system, thus causing downtime. The software upgrade problem is to determine which updates to perform without breaking the system (Ben-Basat et al., 2018). Breaking the system might be as a result of *conflicting* packages, *defective* packages or *dependent* packages. Let P denote the set of *packages*. An installation is a set of packages $I \subseteq P$. An installation can be successful depending on whether all dependencies, conflicts and defect constraints are satisfied. Let $p, q \in P$. A *dependency* $p \to q$ is a constraint that means any successful installation that contains the package p must contain q. A *conflict* $\{p, q\}$ implies that any installation that includes both p and q together will fail. Finally, a package p is a *defect* if it cannot be part of any successful installation. Consider that the dependencies, conflicts and defects are not known to the user, the purpose is to learn a model that for any installation can determine whether the installation will succeed or not. Let $P = \{p_1, \cdots, p_n\}$, R be the set of the dependencies, C the set of conflicting packages and D the set of defectives. Let $x_1, \cdots, x_n$ be the variables indicating whether the package participates in the installation or not. This is equivalent to learning decision tree of depth d from membership queries where a membership query is simulated by actually making the installation and getting a status feedback about it and $d = O(|C| + |R| + |D|)$. For the cases where $d < n$, learning decision trees of depth at most d might be an efficient solution.

1.2. The Learning Model

In the *exact learning from membership queries* model (Angluin, 1987), we have a *teacher* and a *learner*. The teacher holds a hidden *target* function or *concept* $f : \{0,1\}^n \to \{0,1\}$ from a fixed class of Boolean functions C that is known to the learner. The target of the learner is to discover the function f. Given an access to the teacher, the learner can ask the teacher *membership queries* by sending it elements $a \in \{0,1\}^n$. As a response, the teacher returns an answer $f(a) \in \{0,1\}$. The main target of the learner is to learn the exact target function f, using a minimum number of membership queries and a small time complexity. We regard the learner as a *learning algorithm* and the teacher as an *oracle* MQ_f that answers membership queries, i.e., $\mathrm{MQ}_f(a) = f(a)$.

Let C and H be two classes of representations of Boolean functions. Let C be a subset of H in a sense that each function in C has a representation in H. We say that a learning algorithm A *exactly learns* C *from* H in time $t(A)$ *with* (or, and *asks*) $q(A)$ membership queries if for every target function $f \in C$, the algorithm A runs in time at most $t(A)$, asks at most $q(A)$ membership queries and outputs a function $h \in H$ that is logically equivalent to f ($f = h$). If such an algorithm exists, then we say that the class C is *learnable from* H in time $t(A)$ *with* $q(A)$ membership queries (or just, queries).

The learning algorithm can be *deterministic* or *randomized*, *adaptive* (AD) or *non-adaptive* (NAD). In the adaptive learning algorithm, the queries might rely on the results of previous ones. On the other hand, in the non-adaptive algorithm, the queries are independent and must not rely on previous results. Therefore, all the queries can be asked in a single parallel step. We say that an adaptive algorithm is an r*-round adaptive* (r-RAD) if it runs in r stages where each stage is non-adaptive. That is, the queries may depend on the answers of the queries in previous stages, but are independent of the answers of the current stage queries.

A randomized algorithm is called a *Monte Carlo* (MC) algorithm if it uses coin tosses and its running time is finite, while its output might be incorrect with a probability of at most δ, that is specified as an input to the algorithm. All of the randomized algorithms regarded herein are Monte Carlo algorithms.

Let DT_d denote the class of decision trees of depth at most d. We say that the class DT_d is *polynomial time learnable* if there is an algorithm that learns DT_d in $poly(2^d, n)$ time and asks $poly(2^d, \log n)$ membership queries. We cannot expect better time or query complexity since just asking one membership query takes $O(n)$ time. As we will see below, the lower bound for the number of queries is $\Omega(2^d \log n)$.

Along the discussion, we will use the notation $\mathrm{OPT}(\mathrm{DT}_d)$ to indicate the optimal number of membership queries needed to learn the class DT_d with unlimited computational power. More precisely, we use the notation $\mathrm{OPT}_{\mathrm{AD}}(\mathrm{DT}_d)$ ($\mathrm{OPT}_{\mathrm{NAD}}(\mathrm{DT}_d)$, $\mathrm{OPT}_{\mathrm{R,AD}}(\mathrm{DT}_d)$, $\mathrm{OPT}_{\mathrm{R,NAD}}(\mathrm{DT}_d)$) to indicate the optimal number of membership queries needed by a deterministic adaptive (resp., deterministic non-adaptive, randomized adaptive with with failure probability of $\delta = 1/2$, randomized non-adaptive with failure probability of $\delta = 1/2$) algorithm to exact-learn the class DT_d from membership queries.

1.3. Known Results

In the following, we give a brief summary of some of the known results of the learnability of the class DT_d. In (Bshouty, 2018; Bshouty and Costa, 2016), it is shown that[1]:

Theorem 1

$$\tilde{O}(2^{2d}) \log n \geq \mathrm{OPT}_{\mathrm{NAD}}(\mathrm{DT}_d) \geq \mathrm{OPT}_{\mathrm{AD}}(\mathrm{DT}_d) \geq \Omega(2^d \log n)$$

and

$$\tilde{O}(2^{2d}) \log n \geq \mathrm{OPT}_{\mathrm{R,NAD}}(\mathrm{DT}_d) \geq \mathrm{OPT}_{\mathrm{R,AD}}(\mathrm{DT}_d) \geq \Omega(2^d \log n).$$

1. $\tilde{O}(T)$ is $O(poly(d) \cdot T)$. We will use this notation throughout this paper. The exact upper bound is $O(d2^{2d} \log n)$.

Deterministic/ Randomized	Algorithm	Query Complexity
Randomized [(1)]	Feldman (2007) and Bshouty and Costa (2016)	$\tilde{O}(2^{2d})\log n$
Randomized	Ours	$\tilde{O}(2^{2d}) + 2^d \log n$
Deterministic[(2)]	Kushilevitz and Mansour (1993), Jackson (1997), and Blum et al. (1995)	$2^{18d+o(d)}\log n$
Deterministic	Ours	$2^{5.83d} + 2^{2d+o(d)}\log n$

Table 1: Query complexity for polynomial time adaptive algorithms

Achieving an upper bound better than $\tilde{O}(2^{2d})\log n$ is an open problem. Table 1 summarizes some known results on learning decision trees. Most of the algorithms for learning decision trees relate to the target function using the Discrete Fourier Transform representation and try to learn its non-zero Fourier coefficients. Feldman (2007) introduced the first randomized *non-adaptive* algorithm that finds the heavy Fourier coefficients. Applying his algorithm to the class DT_d gives a non-adaptive learning algorithm with query complexity $\tilde{O}(2^{2d})\log^2 n$. See (Bshouty, 2018) for the algorithm. Applying the reduction of Bshouty and Costa (2016) to the algorithm of Feldman gives a non-adaptive learning algorithm with query complexity $\tilde{O}(2^{2d})\log n$. We want to draw the reader's attention that any result for the non-adaptive case implies the same result for the adaptive case. This is result (1) in Table 1. Kushilevitz and Mansour (1993), and Jackson (1997), gave a randomized adaptive algorithm for learning decision trees known as KM algorithm. Kushilevitz and Mansour then use the technique of ϵ-biased distribution to derandomize the algorithm. A rough estimation of the query complexity of their algorithm for the class of DT_d was first done in (Bshouty, 2018). In Appendix A, we show that, with the best derandomization results exist today (Ta-Shma, 2017), the query complexity of their deterministic algorithm is $2^{18d+o(d)}\log n$. This is result (2) in Table 1.

1.4. Our Results

In this paper we give two new algorithms. The first algorithm is a randomized adaptive algorithm (two rounds) that asks $\tilde{O}(2^{2d}) + 2^d \log n$ membership queries. In particular, this shows that $\mathrm{OPT}_{\mathrm{R,AD}}(\mathrm{DT}_d) \le \tilde{O}(2^{2d}) + 2^d \log n$. The second algorithm is a deterministic adaptive algorithm that asks $2^{5.83d} + 2^{2d+o(d)}\log n$. See Table 1.

For the randomized adaptive algorithm, we give a new reduction that changes a randomized r-round algorithm A that learns a class C with $Q(n)$ membership queries to a randomized $(r+1)$-round algorithm that learns C with $Q(8V^2) + v\log n$ queries, where V is an upper bound on the number of relevant variables of the functions in C and v is the number of relevant variables of the target function. The class C must be closed under variable projection. We apply this reduction to the randomized nonadaptive (one round) algorithm of Feldman for DT_d. In Feldman's algorithm we have $Q(n) = \tilde{O}(2^{2d})\log n$. Since, for DT_d, $V, v \le 2^d$, we get a two-round algorithm that asks $\tilde{O}(2^{2d}) + 2^d \log n$ queries.

In our reduction, we randomly uniformly project the variables $(x_1, \ldots, x_n)$ of the target function f into $m = 8V^2$ new variables $(y_1, \ldots, y_m)$. Since the number of the relevant variables of the target function is at most V, with high probability all the variables in the target function are projected to distinct variables. Let $g(y)$ be the projected function. We run the algorithm A to learn $g(y)$. Obviously, queries to g can be simulated by queries to f. This takes $Q(8V^2)$ queries. Algorithm A returns a function $h(y)$ equivalent to g. Then, we show, without asking any more membership queries, how to find the relevant variables of $g(y)$ and how to find witnesses for each relevant variable. That is, for each relevant variable y_i in g, we find two assignments $a^{(i)}$ and $b^{(i)}$ that differ only in y_i such that $g(a^{(i)}) \neq g(b^{(i)})$. Moreover, we show that for each relevant variable y_i in g, one can find the corresponding relevant variable in f by running the folklore nonadaptive algorithm of group testing for finding one defective item combined with the witnesses. Each relevant variable can be found with $\log n$ queries. This requires $v \log n$ queries in the second round.

For the deterministic adaptive algorithm, we first find the relevant variables. Then, we learn the decision tree over the relevant variables using the multivariate polynomial representation, (Bshouty and Mansour, 2002), with the derandomization studied in (Bshouty, 2014). To find the relevant variables, we construct a combinatorial block design, that we will call an (n,d)-*universal disjoint set*, in polynomial time. A set $S \subseteq \{0,1,z\}^n$ is called an (n,d)-universal disjoint set if for every $1 \le i_1 < \cdots < i_d \le n$, every $\xi_1,\ldots,\xi_d \in \{0,1\}^n$ and every $1 \le j \le d$ there is an assignment $a \in S$ such that $a_{i_k} = \xi_k$ for all $k \ne j$ and $a_{i_j} = z$. We regard z as a variable. The algorithm finds all the relevant variables in the target function as follows. To find the relevant variables, the algorithm starts with constructing an $(n, 2d+1)$-universal disjoint set S. For any assignment $a \in S$, let $a[x]$ be the assignment a where each entry i that is equal to z is replaced with x_i. We show that for every decision tree f and every relevant variable x_i in f there is an assignment $a \in S$ such that $f(a[x])$ is equal to either x_i or $\bar{x}_i$. Therefore, when $f(a[1^n]|_{x_i\leftarrow 1}) \neq f(a[0^n]|_{x_i \leftarrow 0})$, we can run the algorithm that learns one literal to learn a relevant variable in f. To learn $f(a[x])$, we need $O(\log n)$ queries. This gives all the relevant variables of f in $|S| + O(v \log n)$ queries where v is the number of relevant variables in f.

In addition, in this paper we give a polynomial time algorithm that constructs an $(n, 2d+1)$-universal disjoint set S of size $|S| = 2^{2d+o(d)} \log n$. This size is optimal up to $2^{o(d)}$. Hence, the total number of queries for the first stage is $2^{2d+o(d)} \log n$.

In the second stage, the algorithm learns the target function over the relevant variables that are found in the first stage. We start by showing that DT_d can be represented as $\mathrm{MP}_{d,3^d}$; multivariate polynomial over the field F_2 with monomials of size (number of variables in the monomial) at most d and at most 3^d monomials. To learn $\mathrm{MP}_{d,s}$, $s = 3^d$, over $N \le 2^d$ variables, we first show how to learn one monomial. In order to (deterministically) learn one monomial, we use the deterministic zero test procedure Zero-Test of $\mathrm{MP}_{d,s}$ from (Bshouty, 2014) that asks $Z = 2^{2.66d} s \log^2 N$ queries. The learning algorithm of $\mathrm{MP}_{d,s}$ from (Bshouty and Mansour, 2002) with this procedure gives a deterministic learning algorithm that asks $O(ZNs)$ queries. This solves the problem with query complexity $\tilde{O}(2^{6.83d})$ (for learning DT_d with $N \le 2^d$ variables). In this paper, we give an algorithm that asks $O(Zs \cdot \log N)$ queries that gives query complexity $\tilde{O}(2^{5.83d})$.

To find one monomial, our algorithm uses (N,d)-*sparse all one set* $D \subseteq \{0,1\}^N$. An (N,d)-sparse all one set is a set $D \subseteq \{0,1\}^N$ where for every $1 \le i_1 < \cdots < i_d \le n$ there is an assignment $a \in D$ such that $a_{i_k} = 1$ for all $k = 1,\ldots,d$ and the weight of each $a \in D$ is at most $N(1 - 1/2d)$. We show how to construct such set in polynomial time. Thereafter, we show that for every $f \in \mathrm{DT}_d$, $f \ne 0$, there is an assignment $a \in D$ such that $f(a * x) \neq 0$ where $a * x = (a_1x_1,\ldots,a_Nx_N)$. To find such an assignment, we use the procedure Zero-Test to zero test each $f(a*x)$, for all $a \in D$. Notice that the weight of a is at most $(1 - 1/2d)N$ and therefore, the function $f(a*x)$ contains at most $(1-1/2d)N$ variables. Thus, if we recursively use this procedure, after at most $2d\ln N$ iterations, we get a monomial M in f. Therefore, to find one monomial we need at most $2^{2.66d}s2d\ln N$ queries.

Assuming that we have found t monomials $M_1,\dots,M_t$ of f, we can learn a new one by learning a monomial of $f' = f + M_1 + M_2 + \cdots + M_t$. The function f' is the function f without the monomials $M_1,\dots,M_t$. This is because $M + M = 0$ in the binary field. Obviously, we can simulate queries for f' with queries for f. Since the number of monomials in f is at most $s = 3^d$, the number of queries in the second stage is at most $2^{2.66d}s^2 2d\ln N = \tilde{O}(2^{5.83d})$.

2. Preliminaries

Let n be an integer. We denote $[n] = \{1,\cdots,n\}$. Consider the set of *assignments*, also called *membership queries* or *queries*, $\{0,1\}^n$. A function f is called a *Boolean function* on n variables $\{x_1,\cdots,x_n\}$ if $f : \{0,1\}^n \to \{0,1\}$. We denote by 1^n (0^n) the all one (zero) assignment. For an assignment a, the i-th entry in a is denoted by a_i. For two assignments $a,b \in \{0,1\}^n$, the assignment $a+b$ denotes the bit-wise xor of the assignments a and b. For an assignment $a \in \{0,1\}^n$, $i \in [n]$ and $\xi \in \{0,1\}$, we denote by $a|_{x_i\leftarrow\xi}$ the assignment $(a_1,\cdots,a_{i-1},\xi,a_{i+1},\cdots,a_n)$. We say that the variable x_i is *relevant* in f if there is an assignment $a = (a_1,\cdots,a_n)$ such that $f(a|_{x_i\leftarrow 0}) \neq f(a|_{x_i\leftarrow 1})$. We call such assignment a *witness* assignment for x_i. A Boolean decision tree is defined as follows:

Definition 1 *Let* DT *denote the set of all Boolean* decision tree *formulas. Then, 1)* $f \equiv 0 \in \mathrm{DT}$ *and* $f \equiv 1 \in \mathrm{DT}$ *and 2) if* $f_0, f_1 \in \mathrm{DT}$*, then for all* x_i*,* $f := \overline{x}_i f_0 + x_i f_1 \in \mathrm{DT}$.

An immediate observation from the construction of f, as defined by step 2 in Definition 1, is that the terms of f are disjoint i.e. there is at most one non-zero term for the same assignment. Consequently, the "+" operation can be replaced with the logic "$\vee$". Hence, we can relate to the function f as a disjunction of pairwise disjoint terms.

Every decision tree f can be *modeled* as a binary tree denoted by $T(f)$ or shortly T if f is known from the context. Let $\mathcal{V}$ denote the vertices set of T. If $f \equiv 0$ or $f \equiv 1$, then T is a node labelled with 0 or 1 respectively. Otherwise, by Definition 1, there are some x_i and $f_0, f_1 \in \mathrm{DT}$ such that $f = \overline{x}_i f_0 + x_i f_1$. Then, T has a root node $r \in \mathcal{V}$ labeled with x_i and two outgoing edges. The left edge is labeled by zero and points to the root node of $T(f_0)$. The right edge is labeled by 1 and points to the root node of $T(f_1)$. The *size* of a decision tree T is the number of its leaves. For any tree T we define $depth(T)$ to be the number of the edges of the longest path from the root to a leaf in T. Let T_f denote all the decision tree models of the Boolean function f. Let d be the depth of the decision tree model $T(f)$. We say that $T(f)$ is a *minimal* model for f if $d = \min_{T\in T_f} depth(T)$. The *depth* of a decision tree f is the depth of its minimal model. We denote by DT_d the class of all decision trees of depth at most d.

Given an assignment $a \in \{0,1\}^n$, the decision tree model $T(f)$ defines a computation. The computation traverses a path from the root to a leaf and assigns values to the variables indicated by the inner nodes in the following way. The computation starts from the root node of the tree $T(f)$. When the computation arrives to an inner node labeled by x_i, it assigns the value of the input a_i to x_i. If $a_i = 1$, then the computation proceeds to the right child of the current node. Otherwise, it proceeds to the left child. The computation terminates when a leaf u is reached. The value of the computation is the value indicated by the leaf u.

Lemma 2 *Let $f_1, f_2 \in \mathrm{DT}_d$, then for any Boolean function $g(y_1, y_2)$ we have $g(f_1, f_2) \in \mathrm{DT}_{2d}$.*

Proof We construct the following tree denoted by $[f_1, f_2]_g$. We replace each leaf v of f_1 by the tree f_2. Denote by $w_{u,v}$ the leaf u of the tree f_2 that replaces v in f_1. The label of $w_{u,v}$ is $g(\ell(v), \ell(u))$, where $\ell(v)$ is the label of v in f_1 and $\ell(u)$ is the label of u in f_2. Therefore, any assignment a that reaches the leaf v in f_1 and u in f_2 satisfies $g(f_1, f_2)(a) = g(f_1(a), f_2(a)) = g(\ell(v), \ell(u))$. On the other hand, this assignment reaches the leaf $w_{u,v}$ in $[f_1, f_2]_g$ which has the label $g(\ell(v), \ell(u))$. The result follows since $[f_1, f_2]_g \in \mathrm{DT}_{2d}$. ■

2.1. Multivariate Polynomial Representation of Decision Trees

A *monotone term* or *monomial* is a product of variables. The *size* of a monomial is the number of the variables in it. A *multivariate polynomial* is a linear combination (over the binary field F_2) of distinct monomials. The size of a multivariate polynomial is the number of monomials in it. It is well known that multivariate polynomials have a unique representation so the size is well defined. Let $\mathrm{MP}_{d,s}$ be the set of Boolean multivariate polynomials over the binary field F_2 with at most s monomials each of size at most d. We say that a set $Z \subseteq \{0,1\}^n$ is a *zero test* for $\mathrm{MP}_{d,s}$ if for every $f \in \mathrm{MP}_{d,s}$ and $f \neq 0$, there is an assignment $a \in Z$ such that $f(a) \neq 0$. In (Bshouty, 2014, Lemma 72), it is proved:

Lemma 3 *There is a zero test set $Z \subseteq \{0,1\}^n$ for $\mathrm{MP}_{d,s}$ of size $2^{2.66d} s \log^2 n$ that can be constructed in polynomial time $poly(2^d, n)$.*

Lemma 4 *Let $f \in \mathrm{DT}_d$. Then, f is equivalent to some $h \in \mathrm{MP}_{d,3^d}$.*

Proof The proof is by induction on d. For $d = 0$, the decision tree is either 0 or 1. The size is at most $1 \leq 3^0$. For $d = 1$, the decision tree is either x_i or $x_i + 1$. The size is at most $2 \leq 3^1$. Suppose the hypothesis is true for $d-1$. Let f be a decision tree of depth d and let x_i be the label of its root. Then, $f = x_i f_1 + (x_i + 1) f_2$ where f_1 and f_2 are trees of depth $d - 1$. By the induction hypothesis, f_1, f_2 are equivalent to $h_1, h_2 \in \mathrm{MP}_{d-1,3^{d-1}}$. Since $f = x_i f_1 + x_i f_2 + f_2 = x_i h_1 + x_i h_2 + h_2 \in \mathrm{MP}_{d,3^d}$, the result follows. ■

2.2. Block Designs

In this section we define two block designs that will be used in the deterministic algorithm for learning decision trees.

A set $S \subseteq \{0,1,z\}^n$ is called an (n,d)-*universal disjoint set* if for every $1 \leq i_1 < \cdots < i_d \leq n$, every $\xi_1, \ldots, \xi_d \in \{0,1\}$ and every $1 \leq j \leq d$, there is an assignment $a \in S$ such that $a_{i_k} = \xi_k$ for all $k \neq j$ and $a_{i_j} = z$. We regard z as a variable. Let H be a family of functions $h : [n] \to [q]$. For $d \leq q$, we say that H is (n,q,d)-*perfect hash family* if for every subset $S \subseteq [n]$ of size $|S| = d$ there is a *hash function* $h \in H$ such that $|h(S)| = d$.

Bshouty (2015) shows that:

Lemma 5 *Let q be a power of prime. If $q > 2d^2$, then there is an (n,q,d)-perfect hash family of size*

$$O\left(\frac{d^2 \log n}{\log(q/(2d^2))}\right),$$

that can be constructed in linear time.

An (n,d)-*universal set* is a set of assignments $U \subseteq \{0,1\}^n$ such that for every $1 \le i_1 < \cdots < i_d \le n$ and every $\xi_1,\ldots,\xi_d \in \{0,1\}$ there is an assignment $a \in U$ such that $a_{i_k} = \xi_k$ for all $k = 1,\ldots,d$. The following is proved in (Naor et al., 1995).

Lemma 6 *An $(8d^2,d)$-universal set of size $2^{d+O(\log^2 d)}$ can be constructed in poly$(2^d,n)$ time.*

We now prove:

Lemma 7 *There is an (n,d)-universal disjoint set $S \subseteq \{0,1,z\}^n$ of size $2^{d+O(\log^2 d)} \log n$ that can be constructed in polynomial time.*

Proof Consider the $(8d^2,d)$-universal set U of size $2^{d+O(\log^2 d)}$ from Lemma 6. We define a set $W \subseteq \{0,1,z\}^t$, for $t = 8d^2$, as follows. For each $a \in U$ we add t vectors $a^{(1)},\ldots,a^{(t)}$ to W where $a^{(i)} = a|_{x_i \leftarrow z} = (a_1^{(i)},\ldots,a_{i-1}^{(i)},z,a_{i+1}^{(i)},\ldots,a_t^{(i)})$. Obviously, W is $(8d^2,d)$-universal disjoint set of size $\ell = 2^{d+O(\log^2 d)}$. Let $w^{(1)},\ldots,w^{(\ell)}$ be the vectors in W. Let $4d^2 < q = 2^r \le 8d^2$, and consider an (n,q,d)-perfect hash family H. By Lemma 5, H is of size $O(d^2 \log n)$, and can be constructed in linear time. We use W and H to construct an (n,d)-universal disjoint set S as follows. For every $h \in H$ and every $w \in W$, we add the assignment $[w,h] = (w_{h(1)},\ldots,w_{h(n)})$ to S.

First, the size of S is $2^{d+O(\log^2 d)} \log n$. Moreover, consider $1 \le i_1 < \cdots < i_d \le n$ and any $\xi_1,\ldots,\xi_d \in \{0,1\}$. Since H is an (n,q,d)-perfect hash family, there is a hash function $h \in H$ such that $r_1 = h(i_1),\ldots,r_d = h(i_d) \in [q]$ are distinct. Since W is an $(8d^2,d)$-universal disjoint set and $q \le 8d^2$ for every $1 \le j \le d$, there is an assignment $w \in S$ such that $w_{r_k} = \xi_k$ for all $k \ne j$ and $w_{r_j} = z$. Therefore, $[w,h]_{i_k} = w_{h(i_k)} = w_{r_k} = \xi_k$ and $[w,h]_{i_j} = z$. Thus, S is an (n,d)-universal disjoint set. ■

An (n,d)-*sparse all one set* $D \subseteq \{0,1\}^n$ is a set such that for every $a \in D$, $wt(a) := |\{i | a_i = 1\}| \le n(1-1/(2d))$, and for every $1 \le i_1 < \cdots < i_d \le n$ there is an assignment $a \in D$ such that $a_{i_k} = 1$ for all $k \in [d]$.

Lemma 8 *There is an (n,d)-sparse all one set $D \subseteq \{0,1\}^n$ of size $d+1$ that can be constructed in polynomial time.*

Proof Consider the $d+1$ vectors $a^{(j)} \in \{0,1\}^n$, $j = 0,1,\ldots,d$ where $a_i^{(j)} = 0$ if $i \mod (d+1) = j$ and $a_i^{(j)} = 1$ otherwise. Consider any $1 \le i_1 < i_2 < \cdots < i_d \le n$. Let $j_t = i_t \, mod(d+1)$ for $t \in [d]$. Therefore, $j_1,\cdots,j_d$ take at most d distinct values from $S = \{0,1,\cdots,d\}$. As a result, there is $k \in S$ such that $j_\ell \ne k$ for all $\ell = 1,\ldots,d$. Then, obviously by the construction of D, $a_{i_1}^{(k)} = \cdots = a_{i_d}^{(k)} = 1$. Moreover, for each j, $wt(a^{(j)}) \le n - \lceil n/(d+1) \rceil \le n(1-1/(2d))$. ■

3. Randomized Adaptive Learning

In this section we give a randomized two-round algorithm that learns DT_d in polynomial time with $\tilde{O}(2^{2d}) + 2^d \log n$ queries. First, we give a reduction that changes a randomized r-round learning algorithm to a randomized $(r+1)$-round learning algorithm while reducing the number of queries for classes with small number of relevant variables. We use this reduction for DT_d with Feldman's algorithm to get the above result.

An m-*variable projection* is a function $P : \{x_1, \ldots, x_n\} \to \{y_1, \ldots, y_m\}$ where $\{y_1, \ldots, y_m\}$ are new distinct variables. For $x = (x_1, \ldots, x_n)$ we write $P(x) = (P(x_1), \ldots, P(x_n))$, where for an assignment $a = (a_1, \cdots, a_n)$, $P(x_i)(a) = a_j$ if $P(x_i) = y_j$. Let C be a class of Boolean functions. We say that C is closed under *variable projection* if for every $f \in C$, every integer m and every m-variable projection P we have $f(P(x)) \in C$. We say that C is $t(n)$-*verifiable* (with success probability at least $1-\delta$) if for every $f \in C$ with n variables one can find, in time $t(n)$ (with probability at least $1 - \delta$), the relevant variables of f, and can derive a witness for each relevant variable x_i, i.e. an assignment $a^{(i)} \in \{0,1\}^n$ such that $f(a^{(i)}|_{x_i \leftarrow 1}) \neq f(a^{(i)}|_{x_i \leftarrow 0})$. We denote by $V(f)$ the number of relevant variables of f. Let $V(C) = \max_{f \in C} V(f)$. The reduction is described in Figure 1. Now we can prove:

Theorem 9 *Let C be a class of functions that is closed under variable projection. $m = 8V(C)^2$. Let H be a $t(n)$-verifiable class with success probability at least $15/16$. Let $\mathcal{A}_n$ be a randomized r-round algorithm that learns the class C from H in time $T(n)$ with $Q(n)$ membership queries and success probability at least $15/16$. Then, there is a randomized $(r+1)$-round algorithm that learns C from H in time $O(T(m)+nQ(m)+t(m)+V(f)n\log n)$ with $Q(m)+V(f)(\log n+1)$ queries and success probability at least $3/4$, where f is the target function.*

Proof Let P be a random uniform m-variable projection (step 1 in Figure 1). The algorithm runs $\mathcal{A}_m$ to learn $g(y) = f(P(x))$ (step 2). Since C is closed under variable projection, $g(y) = f(P(x)) \in C$. Therefore, with probability at least $15/16$, $\mathcal{A}_m$ returns a function $h(y)$ equivalent to $g(y)$. In Lemma 11, we show that, with probability at least $15/16$, the relevant variables of f are mapped by P to different y_i. Therefore, with probability at least $7/8$, $h(y)$ is equivalent to $g(y)$ and the relevant variables of f are mapped by P to different y_i. We now proceed assuming such events are true.

Since $g(y) = h(y) \in H$ and H is $t(n)$-verifiable with success probability at least $15/16$, one can find the relevant variables of g and witnesses for each relevant variable in time $t(m)$ and success probability at least $15/16$. That is, for each relevant variable y_i in g, the procedure Find-Witness in step (3), with probability at least $15/16$, finds an assignment $a^{(i)} \in \{0,1\}^m$ and $\xi_i \in \{0,1\}$ such that $g(a^{(i)}|_{y_i \leftarrow \xi_i}) = 0$ and $g(a^{(i)}|_{y_i \leftarrow \bar{\xi}_i}) = 1$. The procedure Find-Witness returns a set Y of all the relevant variables of $g(y)$ and a set A containing the corresponding witnesses of the relevant variables in g.

Notice that, for all $y_\ell \in Y$, $g(a^{(\ell)}|_{y_\ell \leftarrow \xi_\ell + z}) = z$, $z \in \{0,1\}$. Consider some $y_\ell \in Y$ and let $X_\ell := P^{-1}(y_\ell) = \{x_j \mid P(x_j) = y_\ell\}$. That is, all the variables x_j that are mapped to y_ℓ by P. Notice that, if X is the set of relevant variables of f, then for each $y_\ell \in Y$, the set $X \cap P^{-1}(y_\ell)$ contains exactly one variable that maps to y_ℓ. To find this variable, suppose $P(x) = (y_{j_1}, \ldots, y_{j_n})$. Then $g(a^{(\ell)}) = f(a^{(\ell)}_{j_1}, \ldots, a^{(\ell)}_{j_n})$. Since f has exactly one relevant variable in X_ℓ and $g(a^{(\ell)}|_{y_\ell \leftarrow \xi_\ell + z}) = z$, we have $f(\beta^{(\ell)}) = g(a^{(\ell)}|_{y_\ell \leftarrow \xi_\ell + z}) = z$ where $\beta^{(\ell)} = (\beta^{(\ell)}_1, \ldots, \beta^{(\ell)}_n)$, $\beta^{(\ell)}_i = \xi_\ell + z$ if $x_i \in X_\ell$ and $\beta^{(\ell)}_i = a^{(\ell)}_{j_i}$ if $x_i \notin X_\ell$. Therefore, $f(b^{(\ell)}) = x_t$ where $b^{(\ell)}_i = \xi_\ell + x_i$ if $x_i \in X_\ell$ and $b^{(\ell)}_i = a^{(\ell)}_{j_i}$ if $x_i \notin X_\ell$. Therefore, by learning $f(b^{(\ell)})$ we find the relevant variable in X_ℓ. These are steps 4.1-4.3. In Lemma 12, we show how to learn one variable non-adaptively with at most $\log n + 1$ queries. This is the procedure Learn-Variable in step 4.4. It returns k_ℓ where x_{k_ℓ}

is the variable that corresponds to y_ℓ. Finally, we replace each relevant variable $y_\ell \in Y$ in g with x_{k_ℓ} and get the target function (Step 5).

The success probability of the algorithm is at least $13/16 > 3/4$. The time complexity in step 2 is $O(T(m) + nQ(m))$, in step 3 is $t(m)$, and in step 4.4, by Lemma 12, is $O(V(f)n\log n)$. This gives $O(T(m) + nQ(m) + t(m) + V(f)n\log n)$ time complexity. The query complexity is $Q(m)$ in step 2 and $V(f)\lceil \log n\rceil$ in step 4.4. The number of rounds is r rounds for running $\mathcal{A}_m$ and one round for finding the relevant variables in step 4. ■

In Lemma 13, we show that if the class C is closed under variable projection and is learnable from H in time $T(n)$ and $Q(n)$ queries, then it is verifiable in time $t(n) = O((T(n) + Q(n)\tau(n))n\log n)$ where $\tau(n)$ is the time to compute a function in H. In particular, the result of Theorem 9 is true in time $O((T(m)+Q(m)\tau(m))m\log m+nQ(m)+V(f)n\log n)$. In particular, we have:

Corollary 10 *The class* DT_d *is* 2*-round learnable with a randomized algorithm that runs in polynomial time and asks* $\tilde{O}(2^{2d}) + 2^d \log n$ *queries.*

Proof We use Feldman's non-adaptive (one round) randomized algorithm that runs in $T(n) = poly(2^d, n)$ time and asks $Q(n) = O(d^3 2^{2d}\log^2 n)$ queries. In Lemma 25 in Appendix A, we show that the output of Feldman's algorithm is verifiable in time $O(2^{2d}n)$. For DT_d, we have $V(C) = 2^d$. By Theorem 9, we get a two round randomized algorithm that runs in time $poly(2^d, n)$ and asks $O(d^5 2^{2d} + 2^d \log n)$ queries. ■

Learn(C, n)
$\mathcal{A}_n$ is an algorithm that learns C in time $T(n)$ with $Q(n)$ queries
and let $m = 8V(C)^2$.
1) Let P be random uniform m-variable projection.
2) Run $\mathcal{A}_m$ $\quad\backslash *$ Learning $g(y) = f(P(x))$ $*\backslash$
  2.1) For each membership query $a \in \{0,1\}^m$
    ask $\mathrm{MQ}_f(P(x_1)(a), \ldots, P(x_n)(a))$ $\quad\backslash * \; y_i(a) = a_i \; * \backslash$
  2.2) Let h be the output.
3) $(Y, A) \leftarrow$Find-Witness$(h(y))$
4) For each relevant variable $y_\ell \in Y$ and a witness $a^{(\ell)} \in A$ of y_ℓ
  4.1) Define $\xi_\ell = 1$ if $h(a^{(\ell)}|_{y_\ell \leftarrow 1}) = 0$ and $\xi_\ell = 0$ otherwise.
  4.2) Define $X_\ell = P^{-1}(y_\ell)$.
  4.3) Define $b_i^{(\ell)} = \xi_\ell + x_i$ if $x_i \in X_\ell$ and $b_i^{(\ell)} = a_j^{(\ell)}$ if $x_i \notin X_\ell$
  4.4) $k_\ell \leftarrow$Learn-Variable$(f(b^{(\ell)}))$ $\quad\backslash *$ Using MQ_f $* \backslash$
5) For each $y_\ell \in Y$, replace y_ℓ in h with x_{k_ℓ}.
6) Output(h).

Figure 1: A reduction from an r-round algorithm to an $(r + 1)$-round algorithm.

3.1. Proof of the Lemmas

In this subsection we prove the lemmas used in the previous section.

Lemma 11 *Let $1 \le i_1 \le \cdots \le i_k \le n$. For a random ck^2-variable projection P with probability at least $1-1/(2c)$, $P(x_{i_1}),\ldots,P(x_{i_k})$ are distinct.*

Proof The probability that $P(x_{i_j})$ is equal to $P(x_{i_k})$ for some $j \ne k$ is $1/(ck^2)$. Therefore, by the union bound, the probability that $P(x_{i_1}),\ldots,P(x_{i_k})$ are distinct is at least $1-\binom{k}{2}/(ck^2) \ge 1-1/(2c)$. ■

In the following, we give a non-adaptive algorithm that learns one variable (the procedure Learn-Variable). This result is a folklore result in group testing.

Lemma 12 *There is a non-adaptive learning algorithm that asks $\lceil \log n \rceil$ queries and learns the class $C=\{x_1,\ldots,x_n\}$.*

Proof Denote by $b_i(j)$ the i-th bit of the binary representation of j. Define the queries $a^{(i)} = (b_i(0),\ldots,b_i(n-1))$, $i=1,\ldots,\lceil \log n \rceil$. If the target function is x_ℓ, then the vector of answers to the queries is the binary representation of $\ell-1$. ■

Lemma 13 *Let C be a class of functions that is closed under variable projection. Let $\mathcal{A}_n$ be a randomized learning algorithm that learns C from H in time $T(n)$ with $Q(n)$ queries and success probability at least $15/16$. Let $g(y_1,\ldots,y_m) \in C$ be a computable function in time $\tau(m)$. Then, the relevant variables of g and a witness $a^{(i)}$ for each relevant variable y_i in g can be found in time $O((T(m)+Q(m)\tau(m))m\log m)$ and success probability at least $15/16$.*

Proof We first give an algorithm that for any two functions $g_1, g_2 \in C$ in m variables, with success probability at least $1-1/(16m)$, verifies whether they are equivalent, in time $O((T(m)+Q(m)\tau(m))\log m)$. In case they are not equivalent, the algorithm returns an assignment a such that $g_1(a) \ne g_2(a)$. We run the learning algorithm $\mathcal{A}_m$. For each query a, we check whether $g_1(a) = g_2(a)$. If not, then we are done. Otherwise, stop and return $g_1(a)(=g_2(a))$. If for all the membership queries both functions are equal and the algorithm returns h, then $\Pr[g_1 \ne g_2] \le \Pr[g_1 \ne h] + \Pr[g_2 \ne h] \le 1/8$. Running the algorithm $O(\log m)$ times gives failure probability $1/(16m)$. For each $2 \le i \le m$, define $g^{(i)}(y) = g(y_{y_i \leftarrow y_1})$ and $g^{(1)}(y) = g(y_{y_1 \leftarrow y_2})$. Since C is closed under variable projection, we have $g^{(i)} \in C$. Notice that y_i is a relevant variable if and only if $g(y) \ne g^{(i)}(y)$. The above algorithm can check whether $g = g^{(i)}$ for all i with success probability at least $15/16$. In case, for some i, we get an assignment a such that $g^{(i)}(a) \ne g(a)$ then (assume $i>1$)

$$g^{(i)}(a) = g(a_1,\cdots,a_{i-1},a_1,a_{i+1},\cdots,a_n) \ne g(a).$$

This can happen only when $a_1 \ne a_i$ and then the assignment a is a witness for y_i. ■

4. Deterministic Adaptive Learning

In the following, we give a deterministic algorithm for learning DT_d. In subsection 4.1, we show that there is a polynomial time deterministic algorithm that finds all the relevant variables in $2^{2d+O(\log^2 d)}\log n$ queries. In subsection 4.2, we show that the class $\mathrm{MP}_{d,s}$, defined over N variables, can be learned in a deterministic polynomial time with $2^{2.66d}s^2\log^3 N$ queries.

Theorem 14 *The class DT_d is deterministic polynomial time learnable from $\mathrm{MP}_{d,3^d}$ with $2^{5.83d}+2^{2d+O(\log^2 d)}\log n$ queries.*

Proof By Lemma 19, we find the relevant variables in $2^{2d+O(\log^2 d)}\log n$ queries. The number of relevant variables N is bounded by $N\le 2^d$. Since, by Lemma 4, each decision tree of depth at most d has a representation in $\mathrm{MP}_{d,3^d}$, we can learn it as a multivariate polynomial over the N variables. By Theorem 23, this takes $2^{2.66d}s^2\log^3 N = d^3 2^{2.66d}3^{2d}\le 2^{5.83d}$ queries. ■

4.1. Finding the Relevant Variables

In this section we give an algorithm that asks $2^{2d+O(\log^2 d)}\log n$ queries and finds all the relevant variables of $f\in\mathrm{DT}_d$. We first give a folklore procedure Binary-Learn-Variable that takes two assignments $a,b\in\{0,1\}^n$ such that $f(a)\ne f(b)$ and finds a relevant variable x_j of f such that $a_j+b_j=1$ in $\lceil\log n\rceil$ queries. The procedure defines an assignment c that results from flipping $\lfloor wt(a+b)/2\rfloor$ entries in a that differ from b. Then, it finds $f(c)$ with a membership query. We either have $f(a)\ne f(c)$ or $f(b)\ne f(c)$ because, otherwise, $f(a)=f(b)$ and then we get a contradiction. In the first case, a differs from c in $\lfloor wt(a+b)/2\rfloor$ entries, while in the second case, c differs from b in $\lceil wt(a+b)/2\rceil$ entries. Then, we recursively do the above until, at the tth stage, both assignments differ in $s=\lceil wt(a+b)/2^t\rceil=1$ entries. Therefore, the variable that corresponds to this entry is relevant in f. The number of iterations until $s=1$ is at most $\lceil\log n\rceil$.

We remind the reader that for $a\in\{0,1,z\}^n$, $a[x]$ is the assignment a where each entry i that is equal to z is replaced with x_i. For example, for $a=(1,0,z,1,z)$, $a[x]=(1,0,x_3,1,x_5)$.

Lemma 15 *Let $S\subseteq\{0,1,z\}^n$ be an $(n,2d+1)$-universal disjoint set. If $f\in\mathrm{DT}_d$ depends on x_i, then there is an assignment $a\in S$ such that $f(a[x])\in\{x_i,\bar{x}_i\}$.*

Proof If x_i is relevant in f, then $f(x|_{x_i\leftarrow 0})\ne f(x|_{x_i\leftarrow 1})$. Therefore, $f'(x)=f(x|_{x_i\leftarrow 0})+f(x|_{x_i\leftarrow 1})\ne 0$. Since $f(x|_{x_i\leftarrow 0}),f(x|_{x_i\leftarrow 1})\in\mathrm{DT}_d$, by Lemma 2 we have $f'(x)\in\mathrm{DT}_{2d}$. Let $x_{i_1}\overset{\xi_1}{\to}x_{i_2}\overset{\xi_2}{\to}\cdots\to x_{i_{d'}}\overset{\xi_{d'}}{\to}1$, $d'\le 2d$ be a path in the decision tree representation of f'. Such path exists since $f'\in\mathrm{DT}_{2d}$ and $f'\ne 0$. Obviously, $i\notin\{i_1,\ldots,i_{d'}\}$ since x_i is not relevant in f'. By the definition of S, there is an assignment $a\in S$ such that $a_{i_1}=\xi_1,\ldots,a_{i_{d'}}=\xi_{d'}$ and $a_i=z$. Hence, $f'(a)=1$ and $f'(a[x])=1$. Therefore, either $f(a[x]|_{x_i\leftarrow 0})=1$ and $f(a[x]|_{x_i\leftarrow 1})=0$, or $f(a[x]|_{x_i\leftarrow 0})=0$ and $f(a[x]|_{x_i\leftarrow 1})=1$. As a result, we get $f(a[x])=x_i f(a[x]|_{x_i\leftarrow 1})+\bar{x}_i f(a[x]|_{x_i\leftarrow 0})\in\{x_i,\bar{x}_i\}$. ■

Figure 2 describes the algorithm. In step 1, the algorithm constructs an $(n,2d+1)$-universal disjoint set S. In step 2, it defines an empty set R that eventually, will contain all the relevant variables. Then, in step 3, for every $a\in S$, it defines a new assignment b that is derived from a by assigning to zero all $a_i=z$ in a that indicate a known relevant variable $x_i\in R$. Thereafter, if $f(b[1^n])\ne f(b[0^n])$, the algorithm runs Binary-Learn-Variable$(f,b[1^n],b[0^n])$. The procedure Binary-Learn-Variable satisfies the following:

Find-Relevant(C, n)
1) Construct an $(n, 2d+1)$-universal disjoint set S.
2) $R \leftarrow \emptyset$
3) For every $a \in S$
    3.1) For $i = 1$ to n
        if $a_i = z$ and $x_i \in R$ then $b \leftarrow a|_{x_i \leftarrow 0}$
    3.2) If $f(b[1^n]) \neq f(b[0^n])$ then
        $\ell \leftarrow$Binary-Learn-Variable$(f, b[1^n], b[0^n])$; $R \leftarrow R \cup \{x_\ell\}$
4) Output(R).

Figure 2: An algorithm that returns all the relevant variables.

Lemma 16 *Let $a \in \{0, 1, z\}^n$. If $f(a[1^n]) \neq f(a[0^n])$ then, the procedure call Binary-Learn-Variable $(f, a[1^n], a[0^n])$ finds, in at most $\lceil \log n \rceil$ queries, a relevant variable x_j that satisfies:*
1) $a[1^n]_j + a[0^n]_j = 1$, i.e., $a_j = z$.
2) x_j is a relevant variable in $f(a[x])$ and therefore in f.

The algorithm adds, in step 3.2, the learned relevant variable to the set R. We prove:

Lemma 17 *The number of times that Binary-Learn-Variable is executed is at most $V(f)$, where $V(f)$ the is the number of the relevant variables of f.*

Proof Notice that after step 3.1, for every $x_i \in R$ we have $a_i = 0$ and hence $b_i = 0$ too. Therefore, for every $x_i \in R$, we have $b[1^n]_i = b[0^n]_i$. By Lemma 16, the relevant variable that Binary-Learn-Variable finds is not in R. Also, by Lemma 16, all the variables in R are relevant in f. Thus, Binary-Learn-Variable cannot be executed more than the number of relevant variables of f i.e. $V(f)$. ■

We now show:

Lemma 18 *The procedure Find-Relevant returns all the relevant variables in f.*

Proof Let x_j be a relevant variable in f. By Lemma 15, there is an assignment $a \in S$ such that $f(a[x]) \in \{\bar{x_j}, x_j\}$. If $x_j \in R$ when the algorithm reaches a, then we are done. Otherwise, if $x_j \notin R$, then, the algorithm defines a new assignment b from a such that for each $x_i \in R$ and $a_i = z$ we have $b_i = 0$, otherwise, $b_i = a_i$. In particular, $b_j = z$. Therefore, $f(b[x]) \in \{\bar{x_j}, x_j\}$. This is true because, if $c \in \{0, 1, z\}^n$ and $f(c[x]) = x_j$ (or $\bar{x}_j$), then for any $k \neq j$ where $c_k = z$, $f(c|_{x_k \leftarrow 0}[x]) = f(c[x]|_{x_k \leftarrow 0}) = x_j$ (or $\bar{x}_j$). Therefore, $f(b[1^n]) \neq f(b[0^n])$ and the algorithm runs Binary-Learn-Variable$(f, b[1^n], b[0^n])$. By Lemma 16, the procedure Binary-Learn-Variable finds a relevant variable in $f(b[x]) \in \{\bar{x_j}, x_j\}$ and therefore finds x_j. ■

Lemma 19 *The query complexity of Find-Relevant is $2^{2d+O(\log^2 d)} \log n$.*

Proof Steps 3.1-3.2 are executed $|S|$ times and therefore, by Lemma 7, the number of queries asked in the If condition is $2|S| = 2^{2d+O(\log^2 d)} \log n$. Moreover, by Lemma 17, Binary-Learn-Variable is executed $V(f)$ times and, by Lemma 16, in each execution it asks $\lceil \log n \rceil$ queries. Therefore, the number of queries asked in Binary-Learn-Variable is $V(f)\lceil \log n \rceil = O(2^d \log n)$. ■

4.2. Learning MP

In this section we give an algorithm that learns $\mathrm{MP}_{d,s}$ with $2^{2.66d} s^2 \log^3 N$ queries, where N is the number of variables. We first show how to learn one monomial. The algorithm is in Figure 3.

Learn-Monomial(f, N)
1) If Zero-Test(f) =*True* then
    Output("The function is identically zero") and Halt
2) $b \leftarrow 1^N$;
3) While *True* do
    3.1) Construct a $(wt(b), \min(wt(b)-1, d))$-sparse all one set D.
    3.2) For all $a \in D$
        If Zero-Test$(f((a\Delta b) * x))$ =*false* then Goto 3.4
    3.3) Goto 4
    3.4) $b \leftarrow a\Delta b$
4) Output$\left(\bigwedge_{b_i=1} x_i\right)$.

Figure 3: An algorithm that learns one monomial.

In the algorithm, the procedure Zero-Test answers "True" if f is identically zero and "False" otherwise. It uses Lemma 3. Recall that, 1^N is the all one assignment of dimension N, $wt(b)$ is the Hamming weight of b and $a * x = (a_1x_1, \ldots, a_Nx_N)$. For two assignments $b \in \{0,1\}^N$ and $a \in \{0,1\}^{wt(b)}$, the assignment $a\Delta b$ is defined as follows. Let $1 \le i_1 < i_2 < \cdots < i_{wt(b)}$ be all the indices j such that $b_j = 1$. Then $(a\Delta b)_{i_\ell} = a_\ell$ for all $\ell = 1, 2, \ldots, wt(b)$ and $(a\Delta b)_j = 0$ for all $j \notin \{i_1, \ldots, i_{wt(b)}\}$. For example, if $b = (10011101)$ and $a = (10110)$, then $a\Delta b = (a_1 00 a_2 a_3 a_4 0 a_5) = (10001100)$.

Lemma 20 *If $f(b * x) \ne 0$ then,*

1. *If $f((a\Delta b) * x) \ne 0$ for some $a \in D$, then $wt(a\Delta b) \le wt(b)(1 - 1/(2d))$.*
2. *If $f((a\Delta b) * x) = 0$ for all $a \in D$, then $M = \wedge_{b_i=1} x_i$ is a monomial in f.*

Proof Since D is $(wt(b), \min(wt(b)-1, d))$-sparse all one set and $a \in D$, $wt(a\Delta b) = wt(a) \le wt(b)(1 - 1/(2d))$. This implies *1*. To prove *2*., suppose $f((a\Delta b) * x) = 0$ for all $a \in D$. Let $M = x_{i_1} \cdots x_{i_{d'}}$, $d' \le d$ be any minimal size monomial in $f(b * x)$. In particular, $b_{i_1} = \cdots = b_{i_{d'}} = 1$. From the definition of Δ, we have $(a\Delta b)_{i_\ell} = a_{j_\ell}$, $\ell = 1, \ldots, d'$, for some $1 \le j_1 < j_2 < \cdots < j_{d'} \le wt(b)$. We now prove that $wt(b) = d'$. Suppose, for the sake of contradiction, that $wt(b) \ge d' + 1$. Then, $d'' = \min(wt(b) - 1, d) \ge d'$. Since D is $(wt(b), d'')$-sparse all one set, there is an assignment $a \in D$ such that $a_{j_\ell} = 1$ for all $\ell = 1, 2, \ldots, d'$. Thus, $(a\Delta b)_{i_\ell} = 1$ for all $\ell = 1, 2, \ldots, d'$. As a result, $f((a\Delta b) * x)$ will contain the monomial M and therefore is not zero. This is a contradiction. Therefore, $wt(b) = d'$. Since $wt(b) = d'$ and $b_{i_1} = \cdots = b_{i_{d'}} = 1$, we have $\wedge_{b_i=1} x_i = x_{i_1} \cdots x_{i_{d'}} = M$. ■

Lemma 21 *The number of iterations in step 3 is at most* $O(d \log N)$.

Proof At the beginning $b = 1^N$ and, by Lemma 20, at each iteration the weight of b is reduced by a factor of $1 - 1/(2d) \le e^{-1/(2d)}$. This implies the result. ■

Lemma 21 implies that:

Lemma 22 *Algorithm Learn-Monomial deterministically learns one monomial in the function* f *with* $\tilde{O}(2^{2.66d})s \log^3 N$ *queries.*

Proof By Lemma 3, each zero test takes $2^{2.66d} s \log^2 N$ queries. By Lemma 21, the number of iterations in step 3 is at most $O(d \log N)$. ■

We now prove:

Theorem 23 *There is a deterministic algorithm that learns the class* $\mathrm{MP}_{s,d}$ *in* $\tilde{O}(2^{2.66d})s^2 \log^3 N$ *queries.*

Proof Assume we have found t monomials $M_1, \ldots, M_t$ of f we can learn another new monomial by learning a monomial of $f' = f + M_1 + M_2 + \cdots + M_t$. The function f' is just the function f without the monomials in $M_1, \ldots, M_t$. This is because $M + M = 0$ in the binary field. Using Lemma 22, the result follows. ■

References

Hasan Abasi, Nader H. Bshouty, and Hanna Mazzawi. On exact learning monotone DNF from membership queries. *CoRR*, abs/1405.0792, 2014.

Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, page 197, 2002.

Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

Dana Angluin and Jiang Chen. Learning a hidden graph using $o(\log n)$ queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, 2008.

Ran Ben-Basat, Maayan Goldstein, and Itai Segall. Learning software constraints via installation attempts. *CoRR*, abs/1804.08902, 2018.

E. Biglieri and Lszl Gyrfi. *Multiple Access Channels: Theory and Practice.* IOS Press, 2007.

Avrim Blum, Lisa Hellerstein, and Nick Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. *J. Comput. Syst. Sci.*, 50(1):32–40, 1995.

Nader H. Bshouty. Testers and their applications. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 327–352, 2014.

Nader H. Bshouty. Linear time constructions of some d -restriction problems. In *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, pages 74–88, 2015.

Nader H. Bshouty. Exact learning from an honest teacher that answers membership queries. *Theor. Comput. Sci.*, 733:4–43, 2018.

Nader H. Bshouty and Areej Costa. Exact learning of juntas from membership queries. In *Algorithmic Learning Theory - 27th International Conference, ALT 2016, Bari, Italy, October 19-21, 2016, Proceedings*, pages 115–129, 2016.

Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM J. Comput.*, 31(6):1909–1925, 2002.

Nader H. Bshouty, Dana Drachsler-Cohen, Martin T. Vechev, and Eran Yahav. Learning disjunctions of predicates. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 346–369, 2017.

Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

Ding-Zhu Du and Frank K. Hwang. *Combinatorial Group Testing and Its Applications*. World Scientfic Publishing, 1993.

Ding-Zhu Du and Frank K. Hwang. *Pooling Designs And Nonadaptive Group Testing: Important Tools For Dna Sequencing*. World Scientfic Publishing, 2006.

Arkadiusz Z. Dudek, Tomasz Arodz, and Jorge Galvez. Computational methods in developing quantitative structure-activity relationships (qsar): A review. *Combinatorial Chemistry & High Throughput Screening*, 9(3):213–228, 2006.

Vitaly Feldman. Attribute-efficient and non-adaptive learning of parities and DNF expressions. *Journal of Machine Learning Research*, 8:1431–1460, 2007.

Kurt De Grave, Jan Ramon, and Luc De Raedt. Active learning for high throughput screening. In *Discovery Science, 11th International Conference, DS 2008, Budapest, Hungary, October 13-16, 2008. Proceedings*, pages 185–196, 2008.

Vladimir Grebinski and Gregory Kucherov. Reconstructing a hamiltonian cycle by querying the graph: Application to DNA physical mapping. *Discrete Applied Mathematics*, 88(1-3):147–165, 1998.

Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *J. Comput. Syst. Sci.*, 55(3):414–440, 1997.

Ross D. King, Kenneth E. Whelan, Ffion M. Jones, Philip G. K. Reiser, Christopher H. Bryant, Stephen H. Muggleton, Douglas B. Kell, and Stephen G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427, 2004.

Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993.

Gregory A. Landrum, Julie Penzotti, and Santosh Putta. Machine-learning models for combinatorial catalyst discovery. *MRS Proceedings*, 804:JJ11.5, 2003.

Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 213–223, 1990.

Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191, 1995.

Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 238–251, 2017.

Appendix A. KM-algorithm

Along the discussion, we will frequently need to estimate the expected value of random variables using Chernoff bound. Chernoff bound allows us to compute the sample size needed to ensure that the estimation is close to its mean with high probability. To make the analysis easy to follow for the reader, we bring Chernoff bound in the following Lemma.

Lemma 24 *(Chernoff). Let $X_1, \cdots, X_m$ be independent random variables all with mean μ such that for all i, $X_i \in [-1,1]$. Then, for any $\lambda > 0$,*

$$Pr\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \mu\right| \geq \lambda\right] \leq 2e^{-\lambda^2 m/2}. \tag{1}$$

A.1. The Discrete Fouriere Transform - DFT

One of the powerful techniques used in the literature for learning DT_d is the *Discrete Fourier Transform - DFT*(Bshouty, 2018). To simplify the analysis, it is more convenient to change the range of the Boolean functions from $\{0,1\}$ to $\{-1,+1\}$, i.e. $g : \{0,1\}^n \to \{-1,+1\}$ by redefining the function g from the standard representation of f according to $g = 1 - 2f$. In this case, the zero-labeled leaves of $T(f)$ will be relabeled by -1 while the others remain untouched. Moreover, Definition 1 still holds for this case too with a minor change that $f \equiv -1$ is a decision tree instead of $f \equiv 0$. Similarily, the "+" sign still indicates arethmetic summation. Therefore, we can relate to f as a real valued function $f : \{0,1\}^n \to \mathbb{R}$. The following is an example of a decision tree of depth 3

$$\begin{aligned} f &= x_2(x_3 \cdot (-1) + \bar{x_3} \cdot 1) + \bar{x_2}(x_1 \cdot 1 + \bar{x_1}(x_3 \cdot 1 + \bar{x_3} \cdot (-1))) \\ &= x_2\bar{x_3} - x_2\bar{x_3} + x_1\bar{x_2} + \bar{x_1}\bar{x_2}x_3 - \bar{x_1}\bar{x_2}\bar{x_3}. \end{aligned} \tag{2}$$

Denote by $\mathbb{E}[f]$ the expected value of $f(x)$ with respect to the uniform distribution on x. Along the discussion, in case the expectation is taken over uniform distribution, we will omit the distribution notation.

Let $\mathcal{F}$ denote the set of real functions on $\{0,1\}^n$. $\mathcal{F}$ is a 2^n-dimensional real vector space with inner product defined as:

$$(f,g) := \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)g(x) = \mathbb{E}[f \cdot g].$$

The *norm* of a function f is defined by

$$||f|| := \sqrt{(f,f)} = \sqrt{\mathbb{E}[f^2]}.$$

For each $a \in \{0,1\}^n$ define the *parity function* $\chi_a(x)$:

$$\chi_a(x) = (-1)^{a^T x} = (-1)^{\sum_{i=1}^{n} a_i x_i}.$$

The functions $\{\chi_a\}_{a\in\{0,1\}^n}$ are an orthonormal basis for $\mathcal{F}$. Hence, for all $f \in \mathcal{F}$ we can write

$$f(x) = \sum_{a\in\{0,1\}^n} \hat{f}(a) \cdot \chi_a(x)$$

for $\hat{f}(a) \in \mathbb{R}$. The coefficients $\hat{f}(a)$ are called *Fourier coefficients* and,

$$\hat{f}(a) = \mathbb{E}\left[f(x)\chi_a(x)\right].$$

By Parsaval's Theorem, we get that $\mathbb{E}[f^2] = \sum_{a\in\{0,1\}^n} \hat{f}^2(a)$. For Boolean functions defined over the range $\{-1,+1\}$ we get that,

$$\mathbb{E}[f^2] = \sum_{a\in\{0,1\}^n} \hat{f}^2(a) = 1. \tag{3}$$

We now show that:

Lemma 25 *Let $f = \sum_{a\in A} \lambda_a \chi_a(x)$ be any function where $\lambda_a \neq 0$ for every $a \in A$. Then, the relevant variables of f can be found in time $O(|A|n)$. Moreover, for each relevant variable x_i, an assignment b such that $f(w|_{x_i\leftarrow 0}) \neq f(w|_{x_i\leftarrow 1})$, i.e., a witness for x_i, can be found in time $O(|A|n)$.*

Proof First, since χ_a are orthonormal basis, $f = 0$ if and only if $A = \varnothing$. Second, to find an assignment w such that $f(w) \neq 0$, we find $\xi_1 \in \{0,1\}$ such that $f(x|_{x_1\leftarrow\xi_1}) \neq 0$ and then recursively do that for $x_2,\ldots,x_n$. This takes $O(|A|n)$ steps.

Since $f = f_1 + (-1)^{x_i} f_2$ where $f_1 = \sum_{a\in A, a_i=0} \lambda_a\chi_a(x)$ and $f_2 = \sum_{a\in A,a_i=1} \lambda_a\chi_a(x|_{x_i\leftarrow 0})$, we have $f(x|_{x_i\leftarrow 0}) = f(x|_{x_i\leftarrow 1})$ if and only if $f_2 = 0$. That is, $\{a \in A | a_i = 1\} = \varnothing$. Therefore, x_i is relevant in f if and only if $a_i = 1$ for some $a \in A$. Thus, the set of relevant variables of f is $X = \cup_{a\in A} \cup_{a_i=1} \{x_i\}$.

To find a witness for x_i we need to find an assignment b such that $f(b|_{x_i\leftarrow 1}) \neq f(b|_{x_i\leftarrow 1})$. That is $f(b|_{x_i\leftarrow 1}) - f(b|_{x_i\leftarrow 1}) \neq 0$. Consider the function $g(x) = f(x|_{x_i\leftarrow 1}) - f(x|_{x_i\leftarrow 1})$. Then we need to find an assignment b such that $g(b) \neq 0$. This can be done as above in $O(|A|n)$ steps. ■

A.2. From Decision Tree to DFT Representation

There is a standard way to transform a Boolean function represented as a decision tree to DFT fomulation. Definition 1 implies that every decision tree of size s and depth d can be written as a sum (in $\mathbb{R}$) of s terms each of size at most d. Formally,

$$f(x_1,\cdots,x_n) = \sum_{i=1}^{s} \alpha_i T_i,$$

such that for all $1 \le i \le s$,

$$T_i = \ell_{i_1}\ell_{i_2}\cdots\ell_{i_{d'}},$$

for $d' \le d$, and $\ell_{i_j} = x_{i_j}$ or $\ell_{i_j} = \overline{x}_{i_j}$. Moreover, the parameter α_i is the value of the leaf in the path defined by the term T_i in $T(f)$.

To formulate f in the DFT representation, it is enough to rewrite each term T_i and then sum the whole transformed terms into one formula. Over the real numbers $\mathbb{R}$, any positive literal x_{i_j} in T_i can be expressed as $(1-(-1)^{x_{i_j}})/2$. In a similar fashion, any negative literal $\overline{x}_{i_j}$ translates to $(1+(-1)^{x_{i_j}})/2$. We apply this translation to all the terms T_i. The result is a DFT representation of f. We demonstrate the process with an example. Let $f \in \mathrm{DT}_3$ be the decision tree described in (2). The function f can be written as,

$$f(x_1, x_2, x_3) = x_2x_3 + \overline{x}_2x_1 + \overline{x}_2\overline{x}_1x_3 - x_2\overline{x}_3 - \overline{x}_1\overline{x}_2\overline{x}_3$$

where the first three terms comply with the positive paths in the decision tree and the other two comply with the negative ones. We start with x_2x_3:

$$\begin{aligned} x_2x_3 = & \left(\tfrac{1-(-1)^{x_2}}{2}\right)\left(\tfrac{1-(-1)^{x_3}}{2}\right) \\ & = \tfrac{1}{4} - \tfrac{(-1)^{x_2}}{4} - \tfrac{(-1)^{x_3}}{4} + \tfrac{(-1)^{x_2+x_3}}{4} \\ & = \tfrac{1}{4}\chi_{0000} - \tfrac{1}{4}\chi_{0100} - \tfrac{1}{4}\chi_{0010} + \tfrac{1}{4}\chi_{0110}. \end{aligned}$$

Similarily, we can rewrite $\overline{x}_2x_1$,

$$\begin{aligned} \overline{x}_2x_1 = & \left(\tfrac{1+(-1)^{x_2}}{2}\right)\left(\tfrac{1-(-1)^{x_1}}{2}\right) \\ & = \tfrac{1}{4} - \tfrac{(-1)^{x_1}}{4} + \tfrac{(-1)^{x_2}}{4} - \tfrac{(-1)^{x_2+x_1}}{4} \\ & = \tfrac{1}{4}\chi_{0000} - \tfrac{1}{4}\chi_{1000} + \tfrac{1}{4}\chi_{0100} - \tfrac{1}{4}\chi_{1100}. \end{aligned}$$

To get the full DFT expression of f, we continue in the same fashion for the other three terms. The construction described above imply the following results:

Lemma 26 *Let T be a term of size d' then,*

1. *T has a DFT representation that contains $2^{d'}$ non-zero coefficients $\hat{f}(a)$.*
2. *Each coefficient $\hat{f}(a)$ has the value $\pm\frac{1}{2^{d'}}$.*
3. *Each nonzero coefficient $\hat{f}(a)$ of T satisfies $wt(a) \le d'$.*

Proof The proof follows directly from the construction of DFT from T. ■

Lemma 27 *Let $f \in DT_d$, then for all $a \in \{0,1\}^n$,*

$$\hat{f}(a) = \frac{k}{2^d}$$

for some $-2^d \le k \le 2^d$.

Proof First notice that by Lemma 26, the Fourier coefficients of each term of size at most d is of the form $k/2^d$. Therefore, the Fourier coefficients of the sum of terms of size at most d is of the same form. The fact that $-2^d \le k \le 2^d$ follows from the fact that $|\hat{f}(a)| \le 1$ which follows from Parsaval (3). ■

Using the fact that any $f \in \mathrm{DT}_d$ has at most 2^d terms each of size at most d we can conclude:

Corollary 28 *Let $f \in \mathrm{DT}_d$, then f has a DFT representation that contains at most 2^{2d} non-zero coefficients $\hat{f}(a)$ each having a value in $\{\pm k/2^d | k \in [2^d]\}$ and Hamming weight $wt(a) \le d$.*

A.3. The KM-algorithm

Kushilevitz and Mansour (1993) and Jackson (1997) gave an adaptive algorithm that finds the non-zero coefficients in $poly(2^d, n)$ time and membership queries. Kushilevitz and Mansour algorithm (KM-algorithm) is a recursive algorithm that is given as an input an access to a membership query oracle MQ_f, confidence δ and a string $\alpha \in \{0,1\}^r$. Initially, KM-algorithm runs on the empty string $\alpha = \varepsilon$. The fundamental idea of the algorithm is based on the fact that for any $\alpha \in \{0,1\}^r$ it can be shown that (Jackson, 1997),

$$F_\alpha \triangleq \sum_{x\in\{0,1\}^{n-r}} \hat{f}(\alpha x)^2 = \mathop{\mathbb{E}}_{y,z\in\{0,1\}^r, x\in\{0,1\}^{n-r}}[f(yx)f(zx)\chi_\alpha(y)\chi_\alpha(z)], \tag{4}$$

where for $y = (y_1, \cdots, y_s)$ and $x = (x_1, \cdots, x_t)$, $yx = (y_1, \cdots, y_s, x_1, \cdots, x_t)$. For Boolean functions $f : \{0,1\}^n \to \{-1,+1\}$, (3) and (4) imply that $F_\alpha \le 1$ for all α where equality holds for $\alpha = \varepsilon$. Furthermore, for $f \in \mathrm{DT}_d$ we prove,

Lemma 29 *Let $f \in \mathrm{DT}_d$. Then, for all $\alpha \in \{0,1\}^r$, $r \le n$ there is $\ell \in [2^{2d}] \cup \{0\}$ such that,*

$$F_\alpha = \frac{\ell}{2^{2d}}$$

for $\ell \in [2^{2d}] \cup \{0\}$.

Proof Follows from Corollary 28 and Parsaval (3). ■

KM-algorithm uses divide and conquer technique with the identity in (4) to find the non-zero coefficients. Firstly, the algorithm computes F_0 and F_1 and initiates $T_1 = \{\xi \in \{0,1\} | F_\xi \neq 0\}$. At some stage, the algorithm holds a set $T_r = \{\alpha \in \{0,1\}^r | F_\alpha \neq 0\}$. For each $\alpha \in T_r$, the algorithm computes $F_{\alpha 0}$ and $F_{\alpha 1}$. Since $F_\alpha = F_{\alpha 0} + F_{\alpha 1} \neq 0$, at least one of the terms is not zero. Therefore, the algorithm defines $T_{r+1} = \{\alpha\xi | \alpha \in T_r, \xi \in \{0,1\}, F_{\alpha\xi} \neq 0\} \neq \emptyset$. Using (4), when $r = n$ we can conclude that $F_\alpha = \hat{f}(\alpha)^2$. That is, on the algorithm's termination, $T_n = \{\alpha \in \{0,1\}^n | \hat{f}(\alpha)^2 \neq 0\}$ includes all the non-zero Fourier coefficients of f. Observe that for $f \in \mathrm{DT}_d$, Corollary 28 implies that f has at most 2^{2d} non-zero coefficients. Hence, the number of elements in T_r is at most 2^{2d} for all r. The following lemma summarizes this observation.

Lemma 30 *(Kushilevitz and Mansour, 1993). Let $f \in \mathrm{DT}_d$. Then, for any $1 \le r \le n$, $F_\alpha \neq 0$ for at most 2^{2d} strings $\alpha \in \{0,1\}^r$.*

The immediate result of Lemma 30 is that KM-algorithm computes F_α at most $n \cdot 2^{2d}$ times. Furthermore, from Chernoff bound it follows that:

Lemma 31 *Let $f \in \mathrm{DT}_d$. Let $\alpha \in \{0,1\}^r$ and $1 \le r \le n$. There is an algorithm $\mathcal{A}$ that with probability at least $1-\delta$ exactly computes the value of F_α. Moreover, the algorithm asks $O\left(2^{4d} \cdot \log(\frac{1}{\delta})\right)$ membership queries and runs in time $O\left(2^{4d} \cdot n \cdot \log(\frac{1}{\delta})\right)$.*

Proof Let $\alpha \in \{0,1\}^r$. Chernoff bound can be used to calculate the expectation from (4). Let $W_1, \cdots, W_m$ be m random variables such that $W_i = f(y^{(i)}x^{(i)})f(z^{(i)}x^{(i)})\chi_\alpha(y^{(i)})\chi_\alpha(z^{(i)}) \in$

$\{-1, 1\}$ for random uniform $y^{(i)}, z^{(i)} \in \{0, 1\}^r$ and $x^{(i)} \in \{0, 1\}^{n-r}$. Using Chernoff bound with $\lambda = \frac{1}{2^{2d+1}}$ in (1) gives,

$$\Pr\left[\left|\frac{1}{m}\sum_{i=1}^{m} W_i - F_\alpha\right| \geq \frac{1}{2^{2d+1}}\right] \leq 2e^{-m/2^{4d+1}} < \delta.$$

By choosing $m = O(2^{4d} \cdot \log(1/\delta))$ it follows that with probability at least $1 - \delta$,

$$\left|\frac{1}{m}\sum_{i=1}^{m} W_i - F_\alpha\right| < \frac{1}{2^{2d+1}}.$$

Furthermore, Lemma 29 states that the possible values of F_α vary in steps of $1/2^{2d}$. Since the error is smaller than $\frac{1}{2} \cdot \frac{1}{2^{2d}}$, the exact value of F_α can be found by rounding to the nearest number of the form $\ell/2^{2d}$ where $\ell \in [2^{2d}] \cup \{0\}$.

Each membership query requires generating $O(n)$ random bits, therefore the time complexity is $O(2^{4d} \cdot n \cdot \log(1/\delta))$. ■

From Lemma 31, it can be concluded that,

Theorem 32 *(Kushilevitz and Mansour, 1993). There is an adaptive Monte Carlo learning algorithm $\mathcal{B}$, that with probability at least $1 - \delta$ exactly learns DT_d. Moreover, the algorithm runs in time $O(2^{6d} \cdot n^2 \cdot (\log(n/\delta) + d))$ and asks $O(2^{6d} \cdot n \cdot (\log(n/\delta) + d))$ membership queries.*

Proof As mentioned before, the algorithm computes F_α at most $n \cdot 2^{2d}$ times. By the union bound, to get a failure probability of at most δ, we need to evaluate each F_α with a failure probability $\delta/(n \cdot 2^{2d})$ using algorithm $\mathcal{A}$ from Lemma 31. This implies that each evaluation of F_α requires $O(2^{4d} \cdot (\log(n/\delta) + d))$ membership queries and runs in time $O(2^{4d} \cdot n \cdot (\log(n/\delta) + d))$. Since algorithm $\mathcal{B}$ invokes $\mathcal{A}$ at most $n \cdot 2^{2d}$ times, the theorem follows.

■

A.4. Derandomization

In this subsection, we describe KM-algorithm derandomization in details. We start by giving some preliminaries needed for the analysis.

A.4.1. PRELIMINARIES

Kushilevitz and Mansour use the method of λ-*bias distributions* to derandomize their algorithm. The following definitions are useful to review this derandomization method.

For any set R we will use the notation $\mathbb{E}_R[\cdot]$ to denote $\mathbb{E}_{x \in R}[\cdot]$ with the uniform distribution over the elements of R. Let $S = \{0, 1\}^n$ and let $S' \subseteq S$,

Definition 2 *We say that S' is a d-*wise independent set *if for all uniformly random $x \in S'$ and all $1 \leq i_1 < \cdots < i_d \leq n$ and all $\xi_1, \cdots, \xi_d$ we have*

$$\Pr\left[x_{i_1} = \xi_1, \cdots, x_{i_d} = \xi_d\right] = \frac{1}{2^d}.$$

Alon et al. (1986) showed:

Lemma 33 *Let* $w = \lceil \log n \rceil$, $2s+1 \geq d$ *and* $m = ws+1$. *Then, there is an* $m \times n$ *matrix* M *over the field* F_2 *that can be constructed in* $poly(n)$ *time such that* $S' = \{xM | x \in \{0,1\}^m\}$ *is* d*-wise independent set.*

Definition 3 *We say that* S' *is* λ-biased with respect to linear tests *if for all* $\alpha \in \{0,1\}^n \backslash \{0^n\}$ *it holds that,*

$$\left| \mathop{\mathbb{E}}_{S'}[\chi_\alpha(x)] \right| \leq \lambda.$$

Moreover, Ta-Shma (2017) proves that:

Lemma 34 *There is a* λ*-biased with respect to linear tests* S' *of size*

$$\frac{n}{\lambda^{2+o(1)}}$$

that can be constructed in polynomial time $poly(n, 1/\lambda)$.

Definition 4 *We say that* S' *is* λ-biased with respect to linear tests of size at most d *if for all* $\alpha \in \{0,1\}^n \backslash \{0^n\}$ *such that* $wt(\alpha) \leq d$ *it holds that,*

$$\left| \mathop{\mathbb{E}}_{S'}[\chi_\alpha(x)] \right| \leq \lambda.$$

Naor and Naor (1990) prove that:

Lemma 35 *Given a matrix* M *as in Lemma 33 of size* $m \times n$ *and a* λ*-biased with respect to linear tests set* $\hat{S} \subseteq \{0,1\}^m$, *one can, in polynomial time, construct a* λ*-biased set with respect to linear tests of size at most* d *of size* $|\hat{S}|$.

Taking $s = \lceil (d-1)/2 \rceil$ in Lemma 33, we get $m = O(d \log n)$ and by Lemma 34 and 35 we conclude that:

Lemma 36 *There is a* λ*-biased with respect to linear tests of size at most* d, S' *of size*

$$O\left(\frac{d \log n}{\lambda^{2+o(1)}} \right)$$

that can be constructed in polynomial time $poly(n, 1/\lambda)$.

A.4.2. KM-ALGORITHM DERANDOMIZATION VIA λ-BIAS SAMPLE SPACES

The appropriateness of algorithm derandomization via λ-bias spaces requires showing that the output of the algorithm when its coin tosses are chosen from a uniform distribution, and the output of the algorithm when its coin tosses are chosen from λ-bias distribution are very similar. If this holds, a deterministic version of KM algorithm will replace the Chernoff-based computation of F_α by taking expectation over all elements of the λ-bias sample space. Consequently, a polynomially-small bias sample space is a mandatory requirement. To show "similarity" between the randomized algorithm and the deterministic one, we start with the following lemmas. By Parsaval's Theorem we have:

Lemma 37 *Let $f : S \to \mathbb{R}$ and $S = \{0,1\}^n$. Then,*

$$\mathbb{E}_{x\in S}[f(x)^2] = \sum_{a\in S} \hat{f}(a)^2 .$$

Definition 5 *Let f be any Boolean function. L_1-norm of f is defined as,*

$$L_1(f) \triangleq \sum_{z} |\hat{f}(z)|.$$

By subadditivity of the absolute value we have $L_1(f+g) \le L_1(f) + L_1(g)$ and by Lemma 26, for any term T, $L_1(T) = 1$. Since each $f \in \mathrm{DT}_d$ is a sum of at most 2^d terms we have:

Lemma 38 *Let $f \in \mathrm{DT}_d$. Then, $L_1(f) \le 2^d$.*

Lemma 39 *Let $f : S \to \mathbb{R}$ for $S = \{0,1\}^n$, and $S' \subseteq S$. Let*

$$\lambda = \max_{\substack{a,b\in S\\ a\neq b\\ \hat{f}(a),\hat{f}(b)\neq 0}} |\mathbb{E}_{S'}[\chi_{a+b}(x)]|, \tag{5}$$

then,

$$|\mathop{\mathbb{E}}_{S'}[f(x)^2] - \mathop{\mathbb{E}}_{S}[f(x)^2]| \le (L_1(f)^2 - \mathop{\mathbb{E}}_{S'}[f(x)^2])\lambda \le L_1(f)^2\lambda.$$

Proof Using Lemma 37 we have,

$$\begin{aligned}
\mathop{\mathbb{E}}_{S'}[f(x)^2] &= \mathop{\mathbb{E}}_{S'}\Big[\Big(\sum_{a\in S}\hat{f}(a)\chi_a(x)\Big)^2\Big] \\
&= \mathop{\mathbb{E}}_{S'}\Big[\sum_{a,b\in S}\hat{f}(a)\hat{f}(b)\chi_{a+b}(x)\Big] \\
&= \sum_{a\in S}\hat{f}(a)^2 + \sum_{\substack{a,b\in S\\ a\neq b}}\hat{f}(a)\hat{f}(b)\mathbb{E}_{S'}[\chi_{a+b}(x)] \\
&= \mathop{\mathbb{E}}_{S}[f(x)^2] + \sum_{\substack{a,b\in S\\ a\neq b}}\hat{f}(a)\hat{f}(b)\mathbb{E}_{S'}[\chi_{a+b}(x)].
\end{aligned}$$

Thus we can say,

$$\begin{aligned}
|\mathop{\mathbb{E}}_{S'}[f^2] - \mathop{\mathbb{E}}_{S}[f^2]| &= \left| \sum_{\substack{a,b\in S\\ a\neq b}} \hat{f}(a)\hat{f}(b)\mathbb{E}_{S'}[\chi_{a+b}(x)] \right| \\
&\leq \sum_{\substack{a,b\in S\\ a\neq b}} |\hat{f}(a)|\cdot|\hat{f}(b)|\cdot|\mathbb{E}_{S'}[\chi_{a+b}(x)]| \\
&\leq \max_{\substack{a,b\in S\\ a\neq b\\ \hat{f}(a),\hat{f}(b)\neq 0}} |\mathbb{E}_{S'}[\chi_{a+b}(x)]| \sum_{\substack{a,b\in S\\ a\neq b}} |\hat{f}(a)|\cdot|\hat{f}(b)| \\
&\leq \lambda\cdot\left(\Big(\sum_{a\in S}|\hat{f}(a)|\Big)^2 - \sum_{a\in S}|\hat{f}(a)|^2\right) \\
&\leq \lambda\cdot\Big(L_1(f)^2 - \mathop{\mathbb{E}}_{S}[f^2]\Big) \leq \lambda\cdot L_1(f)^2.
\end{aligned}$$

■

Definition 6 *Let $S_1 = \{0,1\}^k$ and $S_2 = \{0,1\}^{n-k}$ for $0 \leq k \leq n$. Let $S_1' \subseteq S_1$, $\alpha, y \in S_1$ and $x \in S_2$. Define $h_{S_1'} : S_2 \to \mathbb{R}$ such that,*

$$h_{S_1'}(x) \triangleq \mathop{\mathbb{E}}_{y\in S_1'}[f(yx)\chi_\alpha(y)]. \tag{6}$$

When choosing $S_1' = S_1$, we will use the notation $h = h_{S_1}$. In this case h is defined by taking the expectation on uniform distribution over S_1.

Lemma 40 *Let $S = \{0,1\}^n$ and S_1, S_2 as in Definition 6. Let $S_1' \subseteq S_1$. Let $h_{S_1'}$ and $h = h_{S_1}$ be as defined in (6). Then,*

1. $L_1(h) \leq L_1(f)$.
2. $$h_{S_1'} - h = \sum_{a_2\in S_2}\Big(\sum_{\substack{a_1\in S_1\\ a_1\neq\alpha}} \hat{f}(a_1a_2) \mathop{\mathbb{E}}_{y\in S_1}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x).$$

Proof We first prove *1.* Using Definition 6 and the Fourier transform of f and the fact that $\mathbb{E}_{S_1}[\chi_a(y)] = 0$ for all $a \neq 0$ when the expectation is taken over uniform distribution we get,

$$\begin{aligned} h(x) &= \mathop{\mathbb{E}}_{y\in S_1}\left[\sum_{a\in S}\hat{f}(a)\chi_a(yx)\chi_\alpha(y)\right] \\ &= \sum_{a_2\in S_2}\Big(\sum_{a_1\in S_1}\hat{f}(a_1a_2)\mathop{\mathbb{E}}_{y\in S_1}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x) \\ &= \sum_{a_2\in S_2}\hat{f}(\alpha a_2)\chi_{a_2}(x). \end{aligned}$$

The last equation follows because $\mathbb{E}[\chi_{a_1+\alpha}(y)] = 0$ for $a_1 \neq \alpha$. Hence, we get that $\hat{h}(a) = \hat{f}(\alpha a)$. Therefore,

$$L_1(h) = \sum_{a\in S_2}|\hat{h}(a)| = \sum_{a\in S_2}|\hat{f}(\alpha a)| \le \sum_{b\in S}|\hat{f}(b)| = L_1(f).$$

We now prove *2*. Similarily, we can say,

$$\begin{aligned} h_{S_1'}(x) &= \sum_{a_2\in S_2}\Big(\sum_{a_1\in S_1}\hat{f}(a_1a_2)\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x) \\ &= \sum_{a_2\in S_2}\hat{f}(\alpha a_2)\chi_{a_2}(x) + \sum_{a_2\in S_2}\Big(\sum_{\alpha\neq a_1\in S_1}\hat{f}(a_1a_2)\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x) \\ &= h(x) + \sum_{a_2\in S_2}\Big(\sum_{\alpha\neq a_1\in S_1}\hat{f}(a_1a_2)\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x). \end{aligned}$$

■

Lemma 41 *Let $S_1' \subseteq S_1$, and $S_2' \subseteq S_2$ and $h_{S_1'}, h_{S_1} = h$ be as in Definition 6. Let $\lambda_1 = max_{c\in S_1, c\neq 0}|\mathbb{E}_{S_1'}[\chi_c]|$ and $\lambda_2 = max_{c\in S_2, c\neq 0}|\mathbb{E}_{S_2'}[\chi_c]|$ then,*

$$|\mathop{\mathbb{E}}_{S_2'}[h_{S_1'}^2] - \mathop{\mathbb{E}}_{S_2}[h^2]| \le 2L_1(f)\lambda_1 + L_1(f)^2\lambda_2.$$

Proof By the subadditivity of the absolute value function,

$$|\mathop{\mathbb{E}}_{S_2'}[h_{S_1'}^2] - \mathop{\mathbb{E}}_{S_2}[h^2]| \le |\mathop{\mathbb{E}}_{S_2'}[h_{S_1'}^2] - \mathop{\mathbb{E}}_{S_2'}[h^2]| + |\mathop{\mathbb{E}}_{S_2'}[h^2] - \mathop{\mathbb{E}}_{S_2}[h^2]|. \tag{7}$$

Since $-1 \le h_{S_1'} \le 1$ and $-1 \le h \le 1$ we can see that,

$$\begin{aligned} |\mathop{\mathbb{E}}_{S_2'}[h_{S_1'}^2] - \mathop{\mathbb{E}}_{S_2'}[h^2]| &= |\mathop{\mathbb{E}}_{S_2'}[(h_{S_1'} - h)(h_{S_1'} + h)]| \\ &\le \mathop{\mathbb{E}}_{S_2'}[|h_{S_1'} - h| \cdot |h_{S_1'} + h|] \\ &\le 2\mathop{\mathbb{E}}_{S_2'}[|h_{S_1'} - h|]. \end{aligned} \tag{8}$$

By Lemma 40,

$$\begin{aligned}
\mathop{\mathbb{E}}_{x\in S_2'}[|h_{S_1'}-h|] &= \mathop{\mathbb{E}}_{x\in S_2'}\left[\left|\sum_{a_2\in S_2}\Big(\sum_{\alpha\neq a_1\in S_1}\hat{f}(a_1a_2)\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]\Big)\chi_{a_2}(x)\right|\right] \\
&\le \mathop{\mathbb{E}}_{x\in S_2'}\left[\sum_{a_2\in S_2}\Big(\sum_{\alpha\neq a_1\in S_1}|\hat{f}(a_1a_2)|\cdot|\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]|\Big)\cdot 1\right] \\
&= \sum_{a_2\in S_2}\Big(\sum_{\alpha\neq a_1\in S_1}|\hat{f}(a_1a_2)|\cdot|\mathop{\mathbb{E}}_{y\in S_1'}[\chi_{a_1+\alpha}(y)]|\Big) \qquad (9)\\
&\le L_1(f)\cdot\lambda_1.
\end{aligned}$$

Hence, using (8) and (9) we conclude,

$$|\mathop{\mathbb{E}}_{S_2'}[h^2_{S_1'}]-\mathop{\mathbb{E}}_{S_2'}[h^2]|\le 2\cdot L_1(f)\cdot\lambda_1. \qquad (10)$$

On the other hand, by lemmas 39 and 40 we can say that,

$$|\mathop{\mathbb{E}}_{S_2'}[h^2]-\mathop{\mathbb{E}}_{S_2}[h^2]|\le L_1(h)^2\cdot\lambda_2\le L_1(f)^2\cdot\lambda_2. \qquad (11)$$

By combining (7), (10) and (11) the Lemma follows. ■

Using Definition 6 we can rewrite Equation 4:

$$F_\alpha=\mathop{\mathbb{E}}_{x\in S_2}[(\mathop{\mathbb{E}}_{y\in S_1}[f(yx)\chi_\alpha(y)])^2]=\mathop{\mathbb{E}}_{x\in S_2}[h^2_{S_1}]=\mathop{\mathbb{E}}_{x\in S_2}[h^2]. \qquad (12)$$

Derandomization via bias spaces technique is done by choosing polynomially small $S_1'\subseteq S_1$ and $S_2'\subseteq S_2$ such that the expectations needed to evaluate F_α can be done by taking the expectation over the small domains S_1' and S_2'. Let F_α' denote the estimation of F_α derived from (12) by taking the expectations over S_1' and S_2'. That is,

$$F_\alpha'=\mathop{\mathbb{E}}_{x\in S_2'}[(\mathop{\mathbb{E}}_{y\in S_1'}[f(yx)\chi_\alpha(y)])^2]=\mathop{\mathbb{E}}_{x\in S_2'}[h^2_{S_1'}].$$

Let $\Delta(F_\alpha)$ denote the error in evaluating F_α. Namely,

$$\Delta(F_\alpha)\triangleq|F_\alpha-F_\alpha'|.$$

To get good approximation for F_α, we require the value $\Delta(F_\alpha)$ to be small with high probability. Moreover, Lemma 29 implies that requiring $\Delta(F_\alpha)=\frac{1}{2^{2d+1}}$ and rounding the value of F_α' to the closest $\ell/2^{2d}$ gives accurate evaluation of F_α. By lemmas 38 and 41 we choose S_1' and S_2' such that,

$$\Delta(F_\alpha)=|\mathop{\mathbb{E}}_{S_2'}[h^2_{S_1'}]-\mathop{\mathbb{E}}_{S_2}[h^2]|\le 2L_1(f)\lambda_1+L_1(f)^2\lambda_2\le 2^{d+1}\lambda_1+2^{2d}\lambda_2\le\frac{1}{2^{2d+1}},$$

for $\lambda_1=max_{c\in S_1,c\neq 0}|\mathbb{E}_{S_1'}[\chi_c(x)]|$ and $\lambda_2=max_{c\in S_2,c\neq 0}|\mathbb{E}_{S_2'}[\chi_c(x)]|$. The number of membership queries needed in these settings is $|S_1'|\cdot|S_2'|$ for each estimation of F_α. It is easy see that we get optimal sizes when $\lambda_1 = 1/2^{3d+3}$ and $\lambda_2 = 1/2^{4d+2}$. Corollary 28 implies that the non-zero Fourier coefficients of $f \in \mathrm{DT}_d$ have Hamming weight of at most d. All the assignments c in the expectations of χ_c are of the form $c = a + b$ where $\hat{f}(a), \hat{f}(b) \neq 0$ and therefore $wt(c) \leq 2d$. See (5) and (9). Therefore, for λ_1 we need a set that is λ_1-biased with respect to linear tests of size at most $2d$, and for λ_2, a set that is λ_2-biased with respect to linear tests of size at most $2d$. By Lemma 36, such sets can be constructed in $poly(2^d, n)$ time, and are of sizes $d2^{6d+o(d)} \log n$ and $d2^{8d+o(d)} \log n$ respectively. This gives $\tilde{O}(2^{14d+o(d)}) \log^2 n$ queries to exactly find each F_α. As mentioned in the KM-algorithm we need to compute F_α for $n2^{2d}$ α's and therefore

Theorem 42 *There is an adaptive deterministic learning algorithm that exact learns the class* DT_d *in* $poly(2^d, n)$ *time and asks* $\tilde{O}(2^{16d+o(d)})n \cdot \log^2 n$ *membership queries.*

Using the reduction introduced in (Blum et al., 1995) for decision trees we have,

Theorem 43 *There is an adaptive deterministic learning algorithm that exact learns the class* DT_d *in* $poly(2^d, n)$ *time and asks* $\tilde{O}(2^{18d+o(d)}) \cdot \log n$ *membership queries.*