More Efficient PAC-learning of DNF with Membership Queries Under the Uniform Distribution

Nader H. Bshouty Technion

Jeffrey C. Jackson* Duquesne University

Christino Tamon Clarkson University

Corresponding Author:

Jeffrey C. Jackson
Mathematics and Computer Science Department
Duquesne University
600 Forbes Avenue
Pittsburgh, PA 15282-1754
U.S.A.
Email: jackson@mathcs.duq.edu
Tel: 412-396-6466
Fax: 412-396-5197

*This material is based upon work supported by the National Science Foundation under Grants No. CCR-9800029 and CCR-9877079.

Abstract

An efficient algorithm exists for learning disjunctive normal form (DNF) expressions in the uniform-distribution PAC learning model with membership queries [15], but in practice the algorithm can only be applied to small problems. We present several modifications to the algorithm that substantially improve its asymptotic efficiency. First, we show how to significantly improve the time and sample complexity of a key subprogram, resulting in similar improvements in the bounds on the overall DNF algorithm. We also apply known methods to convert the resulting algorithm to an attribute efficient algorithm. Furthermore, we develop techniques for lower bounding the sample size required for PAC learning with membership queries under a fixed distribution and apply this technique to the uniform-distribution DNF learning problem. Finally, we present a learning algorithm for DNF that is attribute efficient in its use of random bits.

Keywords: Probably Approximately Learning; Membership Queries; Disjunctive Normal Form; Uniform Distribution; Fourier Transform.

1 Introduction

Jackson [15] gave the first polynomial-time PAC learning algorithm for DNF with membership queries under the uniform distribution. However, the algorithm's time and sample complexity make it impractical for all but relatively small problems. The algorithm is also not particularly efficient in its use of random bits.

In this paper we significantly improve the time, sample, and random bit complexity of Jackson's Harmonic Sieve. Furthermore, by applying existing techniques, we show that the algorithm can be made attribute efficient. Attribute efficient learning algorithms are standard PAC algorithms with the additional constraint that the sample complexity of the algorithm must be polynomial in $\log n$ (where n represents the total number of attributes) and in all other standard PAC parameters, including the number of attributes that are *relevant* to the target, which we will denote by r.

Specifically, with s representing the size of the DNF expression to be learned and $1-\epsilon$ the desired accuracy of the learned hypothesis, we show that the sample complexity can be reduced from the $\tilde{O}(ns^4/\epsilon^8)$ implicit in Jackson's Harmonic Sieve algorithm to $\tilde{O}(rs^2/\epsilon^4)$. Similarly, the time bound can be reduced from $\tilde{O}(ns^8/\epsilon^{12})$ to $\tilde{O}(rs^4/\epsilon^4)$. Other aspects of the algorithm, such as the form and size of the hypothesis produced, are not adversely affected by these changes. Furthermore, at the expense of producing a more complex hypothesis, the time and sample complexity can be reduced to $\tilde{O}(rs^2/\epsilon^2)$ and $\tilde{O}(rs^4/\epsilon^2)$, respectively, by employing a recent observation of Klivans and Servedio [16].

We obtain our main results by improving on a key Fourier-based subprogram of the Harmonic Sieve. Specifically, the Sieve relies on an algorithm developed by Goldreich and Levin [12] that, given the ability to query a Boolean function f over n Boolean inputs at a polynomial number of points, finds the parity functions (over subsets of the inputs) that are best correlated with f

with respect to the uniform distribution. (The Goldreich-Levin algorithm is often referred to in the learning-theoretic literature as the Kushilevitz-Mansour algorithm, as the latter authors were the first to apply it to larger learning theory questions [17].) As applied in the Sieve, the Goldreich-Levin algorithm runs in time $\tilde{O}(ns^6)$ and sample complexity $\tilde{O}(ns^4)$. The Harmonic Sieve actually needs a slightly modified version of this algorithm and performs additional processing, further increasing the overall bounds on DNF learning.

Levin subsequently developed an alternative algorithm for the parity-learning problem that offers some potential computational benefits [19]. However, a straightforward implementation of the algorithm within the Harmonic Sieve gives a time bound of $\tilde{O}(n^2s^2 + ns^4)$ and sample bound of $\tilde{O}(n^2s^2)$, which while better than Goldreich-Levin in terms of s are worse in terms of n.

We build on Levin's work, developing an algorithm that, when used as a subprogram of the Harmonic Sieve, has time bound and sample complexity $\tilde{O}(ns^2)$ (or $\tilde{O}(rs^2)$ if a relevant attribute detection scheme is applied), improving on both Goldreich-Levin and Levin. This improved algorithm for locating well-correlated parity functions may be of interest beyond our application of it to DNF learning.

We also develop a new technique for finding a lower bound on the sample size needed to learn classes in the PAC-learning model with membership queries. We apply this technique to find a lower bound of roughly $\Omega(s \log r)$ for PAC-learning DNF with membership queries under the uniform distribution. There is thus some gap between this and our upper bound.

Finally, we develop some general derandomization techniques and apply them to obtain a learning algorithm for DNF that is attribute efficient in its use of random bits.

The remainder of the paper is organized as follows. We first give necessary definitions and some background theorems. The key problem analyzed in this paper, which we call the *weak parity problem*, is then defined, and Levin's original algorithm for solving this problem is presented in detail. We next develop two improvements on Levin's algorithm. The performance of the final improved Levin algorithm when it operates as a subprogram of the Harmonic Sieve is then analyzed. Next, we present our lower bound argument and approach to improving randomness efficiency. Finally, suggestions for further work close the paper.

2 Definitions and Notation

Let n be some positive integer and let $[n] = \{1, 2, \ldots, n\}$. We consider Boolean functions of the form $f : \{0,1\}^n \to \{-1, +1\}$, the class $\mathcal{C}_n$ of such functions, and the countable union of such classes $\mathcal{C} = \bigcup_{n \geq 0} \mathcal{C}_n$. In this paper we will focus on the class of DNF expressions. A DNF formula is a disjunction of terms, where each term is a conjunction of literals. A literal is either a variable or its negation. The *size* of a DNF f is the number of terms of f. The class of DNF formulas on n variables consists of Boolean functions on n inputs that can be written as a DNF formulas of size polynomial in n.

For $a \in \{0,1\}^n$, denote the i-th bit of a by a_i. The Hamming weight of a, i.e., the number of ones in a, is denoted $wt(a)$. For $i \in [n]$, the unit vector e_i is the vector of all zeros except for the i-th bit which is one. For $I \subseteq [n]$, the vector e_I denotes the vector of all zeros except at bit positions indexed by I where they are ones. We denote the bitwise exclusive-or between two vectors $a, b \in \{0,1\}^n$ by $a \oplus b$. The dot product $a \cdot b$ is defined as $\sum_{i=1}^{n} a_i b_i$. When dealing with subsets, we identify them with their characteristic vectors, i.e., subsets of $[n]$ with vectors of $\{0,1\}^n$. So for two subsets A, B, the symmetric difference of A and B is denoted by $A \oplus B$.

If $f(n) = O(g(n))$ and $g'(n)$ is $g(n)$ with all logarithmic factors removed, then we write $f(n) = \tilde{O}(g'(n))$. This extends to k-ary functions for $k > 1$ in the obvious way. The function $sign(x)$ returns $+1$ if x is positive and -1 is x is negative.

Let f, h be Boolean functions. We say that h is an *ϵ-approximator* for f under distribution D if $\Pr_D[f \neq h] < \epsilon$. We also use the notation $f \triangle h$ to denote the symmetric difference between f and h, i.e., $\{x : f(x) \neq h(x)\}$. The example oracle for f with respect to D is denoted by $EX(f, D)$. This oracle returns the pair $(x, f(x))$ where x is drawn from $\{0,1\}^n$ according to distribution D. The membership oracle for f is denoted by $MEM(f)$. On input $x \in \{0,1\}^n$, this membership oracle returns $f(x)$. The *Probably Approximately Correct* (PAC) learning model [23] is defined as follows. A class $\mathcal{C}$ of Boolean functions is called *PAC-learnable* if there is an algorithm $\mathcal{A}$ such that for any positive ϵ (accuracy) and δ (confidence), for any $f \in \mathcal{C}$, and for any distribution D, with probability at least $1 - \delta$, the algorithm $\mathcal{A}(EX(f, D), \epsilon, \delta)$ produces an ϵ-approximator for f with respect to D in time polynomial in the size s of f, n, $1/\epsilon$ and $1/\delta$. We call a concept class *weakly PAC-learnable* if it is PAC-learnable with $\epsilon = 1/2 - 1/poly(n, s)$. The ϵ-approximator for f in this case is called a *weak hypothesis* for f. If $\mathcal{C}$ is PAC-learnable by an algorithm $\mathcal{A}$ that uses the membership oracle then $\mathcal{C}$ is said to be *PAC-learnable with membership queries.*

A variable or input x_i to a function f is relevant if $f(a) \neq f(a \oplus e_i)$, for some $a \in \{0,1\}^n$. If there exists a function $\iota(n) = o(n)$ so that $\mathcal{C}$ is PAC-learnable by an algorithm $\mathcal{A}$ that asks $poly(r, s)\iota(n)$ examples and queries, where r is the number of relevant variables of the target function $f \in \mathcal{C}$, then $\mathcal{C}$ is said to be *$\iota(n)$-attribute efficient PAC-learnable* with membership queries. A class is *attribute-efficient PAC-learnable* if it is $\log n$-attribute efficient PAC-learnable. If the algorithm $\mathcal{A}$ outputs an ϵ-approximator h of size $poly(r, s)\iota(n)$, then $\mathcal{C}$ is said to be PAC-learnable attribute efficiently with *small hypothesis.*

The Fourier transform of a Boolean function f is defined as follows. Let $\hat{f}(a) = \mathbf{E}_x[f(x)\chi_a(x)]$ be the *Fourier coefficient* of f at $a \in \{0,1\}^n$, where $\chi_a(x) = (-1)^{a \cdot x}$ and the expectation is taken with respect to the uniform distribution over $\{0,1\}^n$. It is a well-known fact that any Boolean function f can be represented as $f(x) = \sum_a \hat{f}(a)\chi_a(x)$, since the functions χ_a, $a \in \{0,1\}^n$, form an orthonormal basis of real-valued functions over $\{0,1\}^n$. When dealing with a real-valued function $g : \{0,1\}^n \to R$, the notation $|g|$ denotes $\max\{|g(x)| : x \in \{0,1\}^n\}$.

Next we define some notation from coding theory [24]. Let Σ be a finite alphabet of size m. A *code* $\mathcal{C}$ of *word length* n is a subset of Σ^n. The distance between two codewords $x, y \in L$ is defined as $d(x,y) = |\{i \in [n] : x_i \neq y_i\}|$ and the *minimum distance* of $\mathcal{C}$ is defined as

$$d(L) = \min\{d(x,y) | x, y \in \mathcal{C}, x \neq y\}.$$

An q-ary $[n,d]$-code is a code over an alphabet of size q of word length n and minimum distance d. In most cases, the alphabet Σ is a finite field $\mathbb{F}_q$ of size q and the code $\mathcal{C}$ is a linear subspace of $\mathbb{F}_q^n$; in this case, $\mathcal{C}$ is called a q-ary *linear* $[n,k,d]$-code, if $\mathcal{C}$ has dimension k as a subspace. The *generator matrix* G of a q-ary linear $[n,k,d]$-code $\mathcal{C}$ is a $n \times k$ matrix over $\mathbb{F}_q$ whose columns are the basis of $\mathcal{C}$; each codeword in $\mathcal{C}$ is of the form $G \cdot x$, for some $x \in \mathbb{F}_q^k$. Finally, a code $\mathcal{C}$ is called *asymptotically good* if, as $n \to \infty$, both the rate k/n and d/n are bounded away from zero.

3 Sample Size Theorems

We make frequent use of the following two theorems to derive sample sizes sufficient to estimate the mean of a random variable to a specified accuracy with a given confidence level.

Lemma 1 (Hoeffding) *Let $X_1, X_2, \ldots, X_m$ be independent random variables all with mean μ such that for all i, $a \leq X_i \leq b$. Then for any $\lambda > 0$,*

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

It follows that the sample mean of $m = (b-a)^2 \ln(2/\delta)/(2\lambda^2)$ independent random variables having common mean μ will, with probability at least $1-\delta$, be within an additive factor of λ of μ.

Lemma 2 (Bienaymé-Chebyschev) *Let $X_1, X_2, \ldots, X_m$ be pairwise independent random variables all with mean μ and variance σ^2. Then for any $\lambda > 0$,*

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

It follows that the sample mean of $m = \sigma^2/(\delta\lambda^2)$ random variables as described in the lemma will, with probability at least $1-\delta$, be within an additive factor of λ of μ.

4 Weak Parity Learning

While our ultimate goal is to show how to improve the running time and randomization aspects of the Harmonic Sieve algorithm for learning DNF expressions, the core of our speed-up result lies in improving a key procedure of the Sieve. This procedure is used to weakly learn a function $f : \{0,1\}^n \to \{-1,+1\}$ using a hypothesis drawn from the class of parity functions $\mathcal{P} = \{\chi_a, -\chi_a \mid a \in \{0,1\}^n\}$. We will refer to this as the problem of weak parity learning; it is defined formally below.

In this section we will study the weak parity learning problem and present an algorithm that noticeably improves on the previous best asymptotic time bounds of other algorithms for solving this problem. Specifically, letting s represent the smallest number of terms in any DNF representation of the target f, the time bounds for previous algorithms were $\tilde{O}(ns^6)$ [12] and $\tilde{O}(n^2s^2 + ns^4)$ [19]. The time bound for our new algorithm is $\tilde{O}(ns^2)$.

Further below, we will begin our development of a more efficient weak parity algorithm by reviewing Levin's algorithm [19] for this problem. Levin's algorithm has sample and time complexity bounds dependent on n^2 as well as other factors. After describing Levin's algorithm, we will show how to modify it to reduce the bounds to a linear dependence on n. Finally, we give a further modification that improves the time bound's dependence on the DNF-size s of the target function.

Before developing these algorithms, we give a formal definition of the problem to be solved and show how it is related to PAC learning, providing a rationale for our name for this problem.

4.1 The Weak Parity Problem

Definition 1 (θ-Heavy Fourier Coefficient) *A Fourier coefficient $\hat{f}(a)$ of a function $f : \{0,1\}^n \to \{-1,+1\}$ is said to be θ-*heavy *if $|\hat{f}(a)| \geq \theta/c$ for some fixed constant $c \geq 1$ (c will be clear from context when its value is important).*

Definition 2 (Weak Parity Learning) *Given a positive real value θ and a membership oracle $MEM(f)$ for an unknown target function $f : \{0,1\}^n \to \{-1,+1\}$, the* weak parity learning *problem consists of determining whether or not f has a a θ-heavy Fourier coefficient, and if so, finding the index (frequency) of one such coefficient.*

Definition 3 (Weak Parity Algorithm) *A* weak parity algorithm *is an algorithm that, given $MEM(f)$ and θ as above along with a postive real value δ, succeeds with probability at least $1 - \delta$ at solving the weak parity learning problem within a specified run time bound that can depend polynomially on n, θ^{-1}, and $\log \delta^{-1}$.*

Lemma 3 *If $\mathcal{A}(MEM(f), \theta, \delta)$ is a weak parity algorithm then there is an algorithm $\mathcal{A}'(MEM(f), \delta)$ that learns an $\Omega(1/s)$-approximator to f with respect to the uniform distribution, where s is the least number of terms in any DNF representation of the target function f (the DNF-size of f). Furthermore, $\mathcal{A}'$ runs in time bounded by $O(\log^2 s)$ times the running time of $\mathcal{A}(MEM(f), 1/(2s+1), \delta)$. The hypothesis produced by $\mathcal{A}'$ will come from $\mathcal{P}$.*

In short, given a weak parity algorithm as defined above, we can construct an algorithm that weakly learns using $\mathcal{P}$ as the hypothesis class and that has the same run time bound as $\mathcal{A}(MEM(f), 1/(2s+1), \delta)$ to within logarithmic factors. Therefore, every weak parity algorithm as defined above is the essence of an algorithm that weakly learns in the standard PAC sense and that outputs a parity function as its hypothesis. Thus, while we develop weak parity algorithms below that accept an arbitrary threshold value θ as a parameter, our run time bounds for the algorithms will be stated in terms of s by applying the substitution $\theta = 1/O(s)$ and including the $O(\log^2 s)$ factor inherent in our construction of $\mathcal{A}'$.

Proof of Lemma 3: First note that every DNF function f has a Fourier coefficient of magnitude at least $1/(2s+1)$ [15]. We will use this fact to show that any weak parity algorithm $\mathcal{A}(MEM(f), \theta, \delta)$ can be converted to an algorithm $\mathcal{A}'(MEM(f), \delta)$ that with probability at least $1-\delta$ runs within the stated time bound and produces the index a of a Fourier coefficient such that $|\hat{f}(a)| = \Omega(1/s)$. We will then observe that the $(1/s)$-heavy Fourier cofficient produced by $\mathcal{A}'$ corresponds to a parity function that weakly approximates f with respect to the uniform distribution.

We construct $\mathcal{A}'$ from $\mathcal{A}$ as follows. Algorithm $\mathcal{A}'$ consists of running $\mathcal{A}$ repeatedly with different arguments each time, the ith time with arguments $\mathcal{A}(MEM(f), 2^{1-i}, \delta/2^i)$. That is, $\mathcal{A}'$ implements a "guess-and-double" strategy. The algorithm continues to run instances of $\mathcal{A}$ until one of these runs returns a heavy Fourier coefficient. Notice that on run number $\lceil \log(2s+1) \rceil + 1$ (if the algorithm calls $\mathcal{A}$ this many times), $\mathcal{A}$ will be called with its θ parameter assigned a value at most $1/(2s+1)$ and its δ parameter assigned a value at most $\delta/2(2s+1)$. Therefore, this run will with probability at least $1 - \delta/2(2s+1)$ successfully return a $(1/(2s+1))$-heavy Fourier coefficient and run within time $O(\log s)$ times the time bound on $\mathcal{A}(MEM(f), 1/(2s+1), \delta)$, since $\mathcal{A}$ by definition has a time bound logarithmic in $\log \delta^{-1}$.

Furthermore, since the preceding $O(\log s)$ runs of $\mathcal{A}$ use larger parameter values, they will with high probability successfully complete and run within the same time bound as this final run. In fact, our choice of values for the δ parameters in each run guarantees that all of the runs of $\mathcal{A}$, including this final run, will successfully complete within the time bound with probability at least $1-\delta$. Therefore, if one of these earlier runs was to return an index that presumably corresponded to a $(1/(2s+1))$-heavy Fourier coefficient, with probability at least $1-\delta$ this returned index would in fact correspond to a heavy coefficient. Summarizing, this algorithm $\mathcal{A}'$ performs as was claimed above.

To see that a $(1/s)$-heavy Fourier coefficient corresponds to a weakly approximating parity function, we consider the defintion of Fourier coefficients. That is, if a is an index such that $|\hat{f}(a)| \geq 1/s$, then by definition we have that $|\mathbf{E}_{x\sim U_n}[f(x)\chi_a(x)]| \geq 1/s$. Note that, since f and χ_a are both functions mapping to $\{-1,+1\}$, $f(x)\chi_a(x) = 1$ if and only if $f(x) = \chi_a(x)$. Therefore, $\mathbf{E}_{x\sim U_n}[f(x)\chi_a(x)] = 2\mathbf{Pr}_{x\sim U_n}[f(x) = \chi_a(x)] - 1$. So if $|\mathbf{E}_{x\sim U_n}[f(x)\chi_a(x)]| \geq 1/s$ then we will have $\mathbf{Pr}_{x\sim U_n}[f(x) = h(x)] \geq 1/2 + 1/(2s)$ for either $h(x) = \chi_a(x)$ or $h(x) = -\chi_a(x)$. $\square$

4.2 Weak Parity Learning using Levin's Algorithm

In this section we describe Levin's algorithm [19] for solving a problem closely related to the weak parity learning problem. We then show a straightforward way of converting Levin's algorithm to one that explicitly solves the weak parity learning problem as we have defined it.

Levin's algorithm differs from weak parity algorithms as described above in that it returns a short list of Fourier coefficients that with high probability contains a heavy coefficient. To convert this to a weak parity algorithm we must extract a single heavy coefficient from the list. The straightforward method of extracting this coefficient presented in this section is actually somewhat computationally expensive. Some of our later speed-up will come from taking a more sophisticated approach to this part of the problem.

4.2.1 Levin's Algorithm

The specific problem solved by Levin's algorithm is the following: given a membership oracle $MEM(f)$ for some function f and given a positive threshold θ such that there is at least one θ-heavy coefficient, with probability at least $\frac{1}{2}$ return a list of $O(n/\theta^2)$ Fourier indices that contains at least one θ-heavy coefficient. The constant c appearing in the definition of θ-heavy is here taken to be 1.

We begin our development of Levin's algorithm by assuming for the moment that we have guessed that $\hat{f}(a)$ is a heavy coefficient and that we want to verify our guess. Typically, we might perform this verification by drawing a sample X of $x \in \{0,1\}^n$ uniformly at random and computing $\sum_{x \in X} f(x)\chi_a(x)/|X|$. Applying the Hoeffding bound (Lemma 1), it follows that $|X|$ on the order of $\hat{f}^{-2}(a)$ will give a good estimate of $\hat{f}(a)$ with high probability. However, notice that we do not need a completely uniform distribution to produce a coarse estimate with reasonably high probability. In particular, if we draw the examples X from any pairwise independent distribution, then we can apply Chebyshev bounds (Lemma 2) and get that for $|X| \geq 2n/\hat{f}^2(a)$, with probability at least $1 - 1/2n$, $\text{sign}(\sum_{x \in X} f(x)\chi_a(x)) = \text{sign}(\hat{f}(a))$. Thus a polynomial-size sample suffices to find the sign of the coefficient with reasonably high probability, even using a distribution which is only pairwise rather than mutually independent.

One way to generate a pairwise independent distribution is by choosing, for fixed $k > 1$, a random n-by-k 0-1 matrix R and forming the set $Y = \{R \cdot p \mid p \in \{0,1\}^k - \{0^k\}\}$ (the arithmetic in the matrix multiplication is performed modulo 2). This set Y is pairwise independent because each n-bit vector in Y is a linear combination of random vectors, so knowing any one vector Rp gives no information about what any of the remaining vectors might be, even if p is known. Thus if we take $k = \lceil \log_2(1 + (2n/\hat{f}^2(a))) \rceil$ and form the set Y as above then with probability at least

$1 - 1/2n$ over the random draw of R, $\text{sign}(\sum_{x \in Y} f(x)\chi_a(x)) = \text{sign}(\hat{f}(a))$.

We will actually be interested in functions that vary slightly from f. Specifically, for $1 \leq i \leq n$, define $f_i(x) \equiv f(x \oplus e_i)$ and note that

$$\begin{aligned}
\widehat{f_i}(a) &= \mathbf{E}_x[f_i(x)\chi_a(x)] \\
&= \mathbf{E}_x[f(x \oplus e_i)\chi_a(x)] \\
&= \mathbf{E}_x[f(x)\chi_a(x \oplus e_i)] \\
&= \mathbf{E}_x[f(x)\chi_a(x)\chi_a(e_i)] \\
&= (-1)^{a_i}\mathbf{E}_x[f(x)\chi_a(x)] \\
&= (-1)^{a_i}\hat{f}(a).
\end{aligned}$$

(The third equality follows by a change of variables.) Since each of the $\widehat{f_i}(a)$ coefficients has the same magnitude as $\hat{f}(a)$, it follows that for any fixed i and for Y and k as above,

$$\text{sign}\left(\sum_{x \in Y} f(x \oplus e_i)\chi_a(x)\right) = (-1)^{a_i}\text{sign}\left(\hat{f}(a)\right) \tag{1}$$

with probability over the uniform random choice of R of at least $1 - 1/2n$. Therefore, with probability at least $1/2$, (1) holds simultaneously for all i.

Note that instead of summing over x above, we could rewrite this as a sum over $p \in P$, where $P = \{0,1\}^k - \{0^k\}$. Define $f_{R,i}(p) \equiv f((Rp) \oplus e_i)$. Then with probability at least $1/2$ over the choice of R, for all i we have

$$\begin{aligned}(-1)^{a_i}\text{sign}\left(\hat{f}(a)\right) &= \text{sign}\left(\sum_{p\in P} f((Rp)\oplus e_i)\chi_a(Rp)\right)\\ &= \text{sign}\left(\sum_{p\in P} f_{R,i}(p)(-1)^{a^T\cdot R\cdot p}\right)\\ &= \text{sign}\left(\sum_{p\in P} f_{R,i}(p)\chi_{a^TR}(p)\right). \qquad (2)\end{aligned}$$

Now fix $z\in\{0,1\}^k$ and notice that by the definition of the Fourier transform,

$$\widehat{f_{R,i}}(z) = 2^{-k}\sum_{p\in\{0,1\}^k} f_{R,i}(p)\chi_z(p). \qquad (3)$$

If $z = a^TR$ then the sum on the right-hand side of (3) is almost identical to the sum in (2). The only difference is that the 0^k vector has been included in (3).

To handle this off-by-one problem, we make k larger than needed for purposes of applying the Chebyshev bound. Specifically, let $k = \lceil\log_2(1+(2n/\hat{f}^2(a)))\rceil + 1$, so $|Y| = 2^k - 1 > 4n/\hat{f}^2(a)$. This Y is sufficiently large so that it is still the case that with probability at least $1/2$, for all i,

$$(-1)^{a_i}\text{sign}\left(\hat{f}(a)\right) = \text{sign}\left(\sum_{p\in\{0,1\}^k} f_{R,i}(p)\chi_{a^TR}(p)\right).$$

To see this, note that our sample size $|Y|$ is now twice that required by the Chebyshev lemma. Now if the sample is twice as large as required to obtain the correct sign (*i.e.*, twice as large as is required to be within $|\hat{f}(a)|$ of the true mean value), this means that, for fixed i, with probability at least $1-1/2n$

$$\left|\frac{\sum_{p\in P} f_{R,i}(p)\chi_{a^TR}(p)}{|Y|} - \widehat{f_i}(a)\right| \leq \frac{|\widehat{f_i}(a)|}{\sqrt{2}}. \qquad (4)$$

Also, based on the bound on $|Y|$ given above, for all $n > 0$ and any fixed i we have $1/|Y| < |\widehat{f_i}(a)|/4$. It follows that with probability at least $1 - 1/2n$ the sign of the sum in (4) will agree with the sign of $\widehat{f_i}(a)$ even if an additional term with incorrect sign is added to the sum. Therefore, for $z = a^T R$, with probability at least $1/2$ over the choice of R, $(-1)^{a_i}\text{sign}(\hat{f}(a)) = \text{sign}(\widehat{f_{R,i}}(z))$ holds for all i.

Up until now we have been assuming that the index a of a heavy Fourier coefficient of f is known. We're now ready to show how to use the observations above to, with probability at least $1/2$, find a list of coefficients containing such an a.

First, notice that each of the n functions $f_{R,i}$ is a function on k bits, so the entire truth table for each of these functions is of size $2^k \leq 4 + (8n/\hat{f}^2(a))$. This is polynomial in $\hat{f}^{-1}(a)$, where $\hat{f}(a)$ is of magnitude at least θ, since $\hat{f}(a)$ is assumed to be a θ-heavy Fourier coefficient. Therefore, with constant probability we can efficiently produce a list of length 2^k containing the index of a heavy Fourier coefficient as follows. First, select R at random and compute exactly the complete Fourier transforms of each of the n functions $f_{R,i}$. If the Fast Fourier Transform algorithm is used (see, *e.g.*, [1]) then each of these transforms can be computed in time $k2^k$. Next, we turn this collection of n Fourier transforms on k-bit functions into a two dimensional 2^k-by-n table, each column consisting of one of the Fourier transforms. Each row of this table is then converted to an n-bit Fourier index by mapping each value in a row to 0 if the value is positive and 1 otherwise. Then, based on our earlier analysis, with probability at least $1/2$ the row corresponding to $a^T R$ contains the index a if $\hat{f}(a) > 0$ and the ones complement of a otherwise.

Figure 1 shows Levin's algorithm for finding a set containing a θ-heavy Fourier coefficient with probability at least $1/2$. One minor concern addressed in the given implementation of the algorithm is the time required to flip an input bit, *i.e.*, the time required for the operation $x \oplus e_i$. For consistency with our subsequent algorithms, we assume in our presentation of Levin's algorithm that this operation will be performed as part of each membership query, and therefore assume

Input: Membership oracle $MEM(f)(x,i)$ that given x and i returns $f(x \oplus e_i)$; number of input bits n; threshold $0 < \theta < 1$ such that there is at least one Fourier coefficient $\hat{f}(a)$ such that $|\hat{f}(a)| \geq \theta$
Output: With probability at least 1/2 return set containing n-vector a such that $|\hat{f}(a)| \geq \theta$

1. Define $k \equiv \lceil \log_2(1 + (2n/\theta^2)) \rceil + 1$
2. Choose n by k matrix R by uniformly choosing from $\{0,1\}$ for each entry of R
3. Generate the set Y of n-vectors $\{R \cdot p \mid p \in \{0,1\}^k\}$
4. **for each** $i \in [n]$ **do**
5.   **for each** $R \cdot p \in Y$ **do**
6.   Call $MEM(f)(R \cdot p, i)$ to compute $f((R \cdot p) \oplus e_i)$ (call this $f_{R,i}(p)$)
7.   **end do**
8.   Compute (using FFT) $\widehat{f_{R,i}}$
9.   **for each** $z \in \{0,1\}^k$
10.   Compute $a_{z,i} = \text{sign}\left(\widehat{f_{R,i}}(z)\right)$
11.   **end do**
12. **end do**
13. For $z \in \{0,1\}^k$ define $a_z \equiv (\frac{a_{z,1}+1}{2}, \frac{a_{z,2}+1}{2}, \ldots, \frac{a_{z,n}+1}{2})$
14. **return** $\{a_z, \overline{a_z} \mid z \in \{0,1\}^k\}$

Figure 1: Levin's algorithm.

unit time for this operation. This seems a reasonable assumption, as a simple modification to the membership oracle can accomodate this bit-flipping operation with a constant additional time cost per each access to an input bit.

Specifically, the new membership oracle can be thought of as adding an interrupt handler to the original membership oracle. Each time the original oracle requests the value of an input bit, say bit j, the interrupt handler fires. If the second (bit number) argument to the new oracle is equal to j, then the handler returns to the original oracle the complemented value of bit j. Otherwise, it returns unchanged the value of bit j.

In summary, Levin's algorithm produces a list of $2^{k+1} = O(n/\theta^2)$ vectors in time $\tilde{O}(n^2/\theta^2)$, and with probability at least 1/2 one of these vectors is the index of a θ-heavy Fourier coefficient if such a coefficient exists. We next turn to converting this algorithm to a weak parity algorithm.

Input: Number of input bits n; set S containing n-bit vectors; membership oracle $MEM(f)$; threshold $0 < \theta < 1$; confidence $0 < \delta < 1$
Output: Depends on whether or not S contains an element a such that $|\hat{f}(a)| \geq \theta/3$:

- If not, with probability at least $1 - \delta$ returns empty set
- If so, with probability at least $1 - \delta$ returns set containing a single such a.

```
Choose set T of 18 ln(16n/(δθ²))/θ² n-bit vectors uniformly at random
for each a ∈ S do
   Compute f̃(a) = (1/|T|) Σ_{x∈T} f(x)χ_a(x)
   if |f̃(a)| ≥ 2θ/3 then
      return {a}
   end if
end do
return ∅
```

Figure 2: Simple testing phase algorithm.

4.2.2 Producing a Weak Parity Algorithm from Levin's Algorithm

If Levin's algorithm is run $\log_2(2/\delta)$ times then with probability at least $1 - \delta/2$ the union of the sets of vectors returned by the runs contains the index of a θ-heavy coefficient, if the target function has such a coefficient. However, a weak parity algorithm as we have defined it must return a single θ-heavy coefficient. In this section we show how to implement a testing phase that post-processes the set produced by Levin's algorithm, producing a single θ-heavy coefficient with high probability if one exists.

The simple testing strategy applied in this section is illustrated in Figure 2. The primary input to the algorithm is a set S representing the union of the sets produced by $\log 2/\delta$ independent runs of Levin's algorithm. The testing algorithm first draws uniformly at random a single set T of vectors to be used as input to the target function. Next, it calculates the sample mean of the function $f\chi_a$ for each index a in S. The size of T is chosen (by application of the Hoeffding bound) such that with probability at least $1 - \delta$ all of these estimates will be within $\theta/3$ of their true mean values. Therefore, if there is at least one index in S representing a θ-heavy coefficient then with probability at least $1-\delta$ some such index will pass the test at line 4. Similarly, with the same probability all indices representing Fourier coefficients b such that $|\hat{f}(b)| < \theta/3$ will fail the test. Therefore, the algorithm performs as claimed in the figure. Note, however, that the coefficient returned is only guaranteed to be θ-heavy in the $c = 3$ sense of the definition of θ-heavy, not in the $c = 1$ sense that Levin's algorithm guarantees.

Overall, then, by following multiple calls to Levin's algorithm with this testing step (using $\delta/2$ as the confidence parameter), we have a weak parity algorithm. Because the size of the set S that is input to the testing algorithm will be of size $\tilde{O}(n/\theta^2)$, the overall algorithm runs in time $\tilde{O}(n^2/\theta^2 + n/\theta^4)$. It can also be verified that the sample complexity is $\tilde{O}(n^2/\theta^2)$.

In the next subsection we improve on this algorithm, essentially reducing the n^2 terms to linear factors of n.

4.3 Improving on Levin's Algorithm

The observation that $\widehat{f_i}(a) = (-1)^{a_i}\hat{f}(a)$ was crucial to the success of Levin's algorithm. Our basic approach to improving on Levin's algorithm is to extend this observation from the case of a single input bit being flipped to multiple bit flips and to use an algorithm based on multiple bit flips to infer the results we would get from the individual bit flips used in Levin's algorithm. By performing this process multiple times, we can employ a voting strategy that reduces the run time dependence on n in the new algorithm.

As an example of the multiple bit flipping strategy, let a be the index of a heavy Fourier coefficient. For any distinct fixed $i, j \in [n]$, the earlier analysis can be extended in a straightforward way to show that

$$\begin{aligned}\hat{f}(a) &= \mathbf{E}_x[f(x \oplus e_{\{i,j\}})\chi_a(x \oplus e_{\{i,j\}})] \\ &= (-1)^{a_i}(-1)^{a_j}\mathbf{E}_x[f(x \oplus e_{\{i,j\}})\chi_a(x)].\end{aligned}$$

(More generally, for any fixed $I \subseteq [n]$, $\hat{f}(a) = \mathbf{E}_x[f(x \oplus e_I)\chi_a(x)] \prod_{i \in I}(-1)^{a_i}$.) Therefore, for pairwise independent Y as before we have that for a given i and j,

$$\mathrm{sign}(\sum_{x \in Y} f(x \oplus e_{\{i,j\}})\chi_a(x)) = (-1)^{a_i}(-1)^{a_j}\mathrm{sign}(\hat{f}(a))$$

with probability at least $1 - 1/2n$. We also have that

$$\mathrm{sign}(\sum_{x \in Y} f(x \oplus e_j)\chi_a(x)) = (-1)^{a_j}\mathrm{sign}(\hat{f}(a))$$

with the same probability. Thus with probability $1 - 1/n$ both of these sums have the correct sign and can be used to solve for the value of a_i (the sign of the product of the two sums gives $(-1)^{a_i}$). This gives a way to determine the value of a_i that is distinct from Levin's original method. We can do something similar using a_k rather than a_j for some $k \neq j$, giving us yet another way to compute a_i's value.

If there was sufficient independence between these different ways of arriving at values of a_i, then we could use many such calculations and take their majority vote to arrive at a good estimate of the value of a_i. The Hoeffding bound would then show that we could tolerate a much smaller probability of success for each of the individual estimates of mean values, specifically constant probability of success rather than $1 - 1/2n$. Furthermore, if this probability was not dependent on n, then working back through the earlier analysis we see that the variable k used in Levin's

Input: Membership oracle $MEM(f)(x, I)$ that given x and I returns $f(x \oplus e_I)$; number of input bits n; threshold $0 < \theta < 1$ such that there is at least one Fourier coefficient $\hat{f}(a)$ such that $|\hat{f}(a)| \geq \theta$
Output: With probability at least 1/4 return set containing n-vector a_z such that $|\hat{f}(a_z)| \geq \theta$

1. Choose constant $0 < c < 1/8$ (different choices will give different performance for different problems)
2. Define $t \equiv \lceil 2\ln(4n)/(1-8c)^2 \rceil$
3. Choose set T consisting of t uniform random n-vectors over $\{0, 1\}$
4. Define $k \equiv \lceil \log_2(1 + 1/c\theta^2) \rceil + 1$
5. Choose n by k matrix R by uniformly choosing from $\{0, 1\}$ for each entry of R
6. Generate the set Y of n-vectors $\{R \cdot p \mid p \in \{0, 1\}^k\}$
7. **for each** $I \in T$ **do**
8.   **for each** $R \cdot p \in Y$ **do**
9.   Call $MEM(f)(R \cdot p, I)$ to compute $f((R \cdot p) \oplus e_I)$ (call this $f_{R,I}(p)$)
10.   **end do**
11.   Compute (using FFT) $\widehat{f_{R,I}}$
12.   **for each** $i \in [n]$ **do**
13.   Compute $f_{R,I \oplus \{i\}}(p)$ as in line 9.
14.   Compute (using FFT) $\widehat{f_{R,I \oplus \{i\}}}$
15.   **end do**
16. **end do**
17. **for each** $z \in \{0, 1\}^k$ **do**
18.   **for each** $i \in [n]$
19.   Compute $a_{z,i} = \text{sign}\left(\sum_{I \in T} \text{sign}\left(\widehat{f_{R,I}}(z) \cdot \widehat{f_{R,I \oplus \{i\}}}(z)\right)\right)$
20.   **end do**
21. **end do**
22. **return** $\{a_z \mid z \in \{0, 1\}^k\}$

Figure 3: Improved Levin algorithm.

algorithm also would not depend on n. This in turn would result in an overall reduction in the run time bound's dependence on n.

These observations form the basis for the algorithm shown in Figure 3. First, notice that for $k = \lceil \log_2(1 + 1/c\theta^2) \rceil$ we have that, for a such that $|\hat{f}(a)| > \theta$, with probability at least $1 - c$ over the choice of R

$$\text{sign}(\sum_{x \in Y} f(x)\chi_a(x)) = \text{sign}(\hat{f}(a)).$$

Furthermore, for any fixed $I \subseteq [n]$, we can similarly apply Chebyshev to $f(x \oplus e_I)$ and get that with the same probability $1 - c$ over uniform random choice of R,

$$\operatorname{sign}\left(\sum_{x \in Y} f(x \oplus e_I)\chi_a(x)\right) = \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j \in I} (-1)^{a_j}. \tag{5}$$

This means that the expected fraction of I's that fail to satisfy (5) for uniform random choice of R is at most c. Therefore, by Markov's inequality, the probability of choosing an R such that a $2c$ or greater fraction of the I's fail to satisfy (5) is at most $1/2$. We will call such an R *bad for a* and all other R's *good for a*. Furthermore, for any fixed $i \in [n]$ and for any R that is good for a, at most a $2c$ fraction of the I's fail to satisfy the following equality:

$$\operatorname{sign}\left(\sum_{x \in Y} f(x \oplus e_I \oplus e_i)\chi_a(x)\right) = \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j \in I \oplus \{i\}} (-1)^{a_j}. \tag{6}$$

This follows because for any fixed R there is a one-to-one correspondence between the I's that fail to satisfy this equality and those that fail to satisfy (5). Therefore, combining (5) and (6), for fixed i, the probability over random choice of R is also at most $1/2$ that for a greater than $4c$ fraction of the I's, either of the following conditions holds (each condition is a conjunction of two relational expressions):

$$\operatorname{sign}\left(\sum_{x \in Y} f(x \oplus e_I)\chi_a(x)\right) \neq \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j \in I} (-1)^{a_j}$$

and

$$\operatorname{sign}\left(\sum_{x \in Y} f(x \oplus e_I \oplus e_i)\chi_a(x)\right) = \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j \in I \oplus \{i\}} (-1)^{a_j},$$

or

$$\operatorname{sign}\left(\sum_{x \in Y} f(x \oplus e_I)\chi_a(x)\right) = \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j \in I} (-1)^{a_j}$$

and

$$\operatorname{sign}\left(\sum_{x\in Y} f(x\oplus e_I\oplus e_i)\chi_a(x)\right) \neq \operatorname{sign}\left(\hat{f}(a)\right) \prod_{j\in I\oplus\{i\}} (-1)^{a_j}.$$

This in turn implies that for fixed i,

$$\Pr_R\left[\Pr_I\left[\operatorname{sign}\left(\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)\cdot\sum_{x\in Y} f(x\oplus e_I\oplus e_i)\chi_a(x)\right)\neq(-1)^{a_i}\right]\geq 4c\right]\leq\frac{1}{2}.$$

So, for fixed i and R's that are good for a, a random choice of I has probability at least $1-4c$ of giving the correct sign for a_i and therefore probability at most $4c$ of giving the incorrect sign. If the correct sign is $+1$, then for a good R and any fixed i,

$$\mathbf{E}_I\left[\operatorname{sign}\left(\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)\cdot\sum_{x\in Y} f(x\oplus e_I\oplus e_i)\chi_a(x)\right)\right]\geq 1-8c.$$

Similarly, a correct sign of -1 gives expected value bounded by $-1+8c$. So by Hoeffding, if we estimate this expected value by taking a sum over t randomly chosen I's, for

$$t\geq\frac{2\ln 4n}{(1-8c)^2}, \tag{7}$$

then the sign of the estimate will be $(-1)^{a_i}$ with probability at least $1-1/2n$. Therefore, for any R good for a, with probability at least $1/2$ the sign estimates of $(-1)^{a_i}$ will be correct simultaneously for all n possible values of i. In this case, we will discover all n bits of the index a of the heavy Fourier coefficient $\hat{f}(a)$. Since we have probability at least $1/2$ of choosing a good R, overall we succeed at finding a with probability at least $1/4$ over the random choice of R and of the t values of I.

One final detail that must be addressed is that while the above analysis is in terms of sums of the form $\sum_{x \in Y}$, the implicit sums computed by the algorithm, using the FFT, effectively include the bit-vector $x = 0^n$. As with the analysis of Levin's orginal algorithm, we can overcome this problem by using a somewhat larger k value than the above analysis would indicate. It is easy to verify that the k given in Figure 3 is sufficient to overcome this off-by-one problem.

This algorithm has sample and time complexity $\tilde{O}(n/\theta^2)$. However, as before, this algorithm is not by itself a weak parity algorithm. To find a single Fourier coefficient that is θ-heavy again requires testing possibly each of the $2^k = O(\theta^{-2})$ coefficients returned by the improved Levin's algorithm, and the simple test phase as given in Figure 2 takes time $\tilde{O}(\theta^{-2})$ per coefficient. So the time complexity of the resulting weak parity algorithm is now $\tilde{O}(n/\theta^2 + \theta^{-4})$. While an improvement over the bound for the previous weak parity algorithm, this is still not as good as the sample complexity bound, which can be seen to be $\tilde{O}(n/\theta^2)$. It would of course be nice to get a similar bound on the time complexity, and in fact this is what we do next.

4.4 Improving the Testing Phase

In this section, we will show how to further modify Levin's algorithm so that it produces a list containing *only* θ-heavy coefficients (in the $c = 3$ sense of θ-heavy), still in time $\tilde{O}(n/\theta^2)$. To convert this to a weak parity algorithm is then simply a matter of choosing one of the elements of the list arbitrarily, so the testing phase is no longer needed. Thus the resulting weak parity algorithm has an overall time bound of $\tilde{O}(n/\theta^2)$.

The basic idea is that we will now "tighten up" our estimates of $\mathbf{E}[f(x \oplus e_I)\chi_a(x)] = \prod_{j \in I}(-1)^{a_j}\hat{f}(a)$. Earlier, we were only interested in getting the *sign* of our estimates of this expectation correct. Now, we will want to obtain estimates of this expectation that are (with high probability) within $\theta/3$ of the true value. Because for each choice of R, $\widehat{f_{R,I}}(a^T R)$ is (approximately) an estimate of $\mathbf{E}[f(x \oplus e_I)\chi_a(x)]$, we can then decide whether or not a given z in the algorithm of Figure 3 corresponds to a θ-heavy a: loosely, if $|\widehat{f_{R,I}}(z)| < 2\theta/3$ then z does not correspond to a θ-heavy a, and we will eliminate the corresponding a_z from the output list of coefficients. Otherwise, the corresponding a_z is θ-heavy, and we leave it in the list.

Actually, what we have just outlined will not quite work, because we cannot afford to estimate the expectation for each value of I to within $\theta/3$ and still reach our desired time bound. Instead, because we are not really interested in the values of the expectation for individual I's but are merely using various I's to obtain an estimate of the magnitude of $\hat{f}(a)$, we can once again use a voting scheme. An argument similar to the earlier one will then show that this voting scheme succeeds within the required time bounds.

Finally, we must also eliminate from the final list of coefficients any "stray" coefficients produced by "bad" choices of R or of a set of I's. That is, it is quite possible that particular choices of R or a set of I's could result in a non-heavy coefficient appearing to be heavy. However, it is much less likely that this same non-heavy coefficient will consistently appear to be heavy for a number of independently chosen R's and sets of I's. Therefore, we can once again employ voting to eliminate these non-heavy coefficients from the final output list.

Formalizing these ideas, we arrive at the algorithm of Figure 4. In what follows we will outline the correctness of this algorithm. First consider two coefficients a and b where a is *fully θ-heavy* ($|\hat{f}(a)| \geq \theta$) and b is *θ-light* ($|\hat{f}(b)| < \theta/3$). Choose $k = \lceil \log_2(1+1/(c\theta^2)) \rceil$ for some constant c, let R be a uniform random n by k matrix as before, and also as before take $Y = \{R \cdot p \mid p \in \{0,1\}^k - \{0^k\}\}$. Then define

$$\Delta_{R,I,d} \equiv \left| \frac{1}{2^k - 1} \sum_{x \in Y} f(x \oplus e_I)\chi_d(x) - \hat{f}(d)\chi_d(e_I) \right|$$

Input: Membership oracle $MEM(f)(x, I)$ that given x and I returns $f(x \oplus e_I)$; number of input bits n; threshold $0 < \theta < 1$ such that there is at least one Fourier coefficient $\hat{f}(a)$ such that $|\hat{f}(a)| \geq \theta$; confidence parameter $0 < \delta < 1$.
Output: With probability at least $1 - \delta$ return set containing all of the n-vectors a such that $|\hat{f}(a)| \geq \theta$ and no n-vectors b such that $|\hat{f}(b)| < \theta/3$

Choose positive real constants c_1, c_2, δ_1, δ_2 such that $\Gamma \equiv 1 + (\delta_1 + \delta_2)(1/c_1 + 1/c_2) - 2(\delta_1 + 1/c_1) - (\delta_2 + 1/c_2)$ is positive, $1/c_1 + 1/c_2 < 1$, and $\delta_1 + \delta_2 < 1$
Choose constant $0 < c < \min(1/18c_1, 1/4c_2)$
Define $k \equiv \lceil \log_2(1 + 1/(c\theta^2)) \rceil + 1$
Define $\lambda \equiv \sqrt{4 + 2\Gamma} - 2$
Define $\ell \equiv$ smallest integer such that $\ell \geq \ln(6\ell/c\delta\theta^2)/2\lambda^2$
Define $t \equiv \lceil 2\ln(2n/\delta_2)/(1 - 4c_2c)^2 \rceil$
Define $m \equiv \lceil 2\ln(2/\delta_1)/(1 - 18c_1c)^2 \rceil$
$L \leftarrow$ empty list
repeat ℓ **times**
    Choose n by k matrix R by uniformly choosing from $\{0,1\}$ for each entry of R
    Generate the set Y of n-vectors $\{R \cdot p \mid p \in \{0,1\}^k\}$
    for each $z \in \{0,1\}^k$ **do** set $V(z)$ to 0
    Choose set T consisting of $\max(m, t)$ uniform random n-vectors over $\{0,1\}$
    for each $I \in T$ **do**
        for each $R \cdot p \in Y$ **do** compute $f_{R,I}(p)$ (using $MEM(f)$)
        Compute (using FFT) $\widehat{f_{R,I}}$
        for each $z \in \{0,1\}^k$ **do** add $V_{I,R}(z)$ to $V(z)$ (see equation (10))
        for each $i \in [n]$ **do**
            for each $R \cdot p \in Y$ **do** compute $f_{R,I\oplus\{i\}}(p)$ (using $MEM(f)$)
            Compute (using FFT) $\widehat{f_{R,I\oplus\{i\}}}$
        end do
    end do
    for each $z \in \{0,1\}^k$ **do**
        if $V(z) > 0$ **then**
            Define each bit $i \in [n]$ of a_z by $a_{z,i} = \text{sign}\left(\sum_{I \in T} \text{sign}\left(\widehat{f_{R,I}}(z) \cdot \widehat{f_{R,I\oplus\{i\}}}(z)\right)\right)$
            if $a_z^T R = z$ **then** add a_z to L
        end if
    end do
end repeat
Define $\Theta \equiv \ell((1 - \delta_1 - \delta_2)(1 - 1/c_1 - 1/c_2) - \lambda)$
return $\{a : a$ appears at least Θ times in $L\}$

Figure 4: An algorithm incorporating the magnitude test.

where d is an n-bit vector. Chebyshev's inequality then gives that for fixed I and any fixed d,

$$\Pr_R[\Delta_{R,I,d} \geq \theta/3] \leq 9c.$$

By a Markov argument as before, this implies that for fixed d and constant $0 < c_1 < 1$,

$$\Pr_R\left[\Pr_I[\Delta_{R,I,d} \geq \theta/3] \geq 9c_1 c\right] \leq \frac{1}{c_1}. \tag{8}$$

Similarly, for constant $0 < c_2 < 1$,

$$\Pr_R\left[\Pr_I[\Delta_{R,I,d} \geq \theta] \geq c_2 c\right] \leq \frac{1}{c_2}. \tag{9}$$

For fixed d and θ, we call R *magnitude good for* d if both $\Pr_I[\Delta_{R,I,d} \geq \theta/3] \leq 9c_1 c$ and $\Pr_I[\Delta_{R,I,d} \geq \theta] \leq c_2 c$. Note that the probability of drawing a magnitude good R for any fixed d is at least $1 - (1/c_1 + 1/c_2)$ by inequalities (8) and (9).

Next, we define a voting procedure, which we would like to produce $+1$ if its k-bit input z corresponds to a fully heavy coefficient and -1 if z corresponds to a light coefficient. We will see below that the following procedure frequently (over choices of R and I) performs in just this way:

$$V_{R,I}(z) = \begin{cases} 1 & \text{if } \left|\widehat{f_{R,I}}(z)\right| \geq \frac{\frac{2(2^k-1)\theta}{3} + \text{sign}(\widehat{f_{R,I}}(z)) f(e_I)}{2^k} \\ -1 & \text{otherwise} \end{cases} \tag{10}$$

Definition 4 *Let $\mathcal{R}$ be the multiset of R's generated on line 10 by a single run of the algorithm. Let $d \in \{0,1\}^n$ be a fully heavy (resp. light) coefficient, let $R \in \mathcal{R}$ be magnitude good for d, and let $z_d \equiv d^T R$. Then the pair (z_d, R) is called a (d,R)-**decisive pair**. For a fully heavy (resp. light) coefficient d, Z_d represents the set of all (d,R)-decisive pairs.*

It is convenient to define a partial function $s : \{0,1\}^n \to \{-1,+1\}$ that produces $+1$ if its input is the index of a fully heavy coefficient, -1 if it's input corresponds to a light coefficient, and is otherwise undefined. Then if d represents a fully heavy or light coefficient, we will say that a (d,R)-decisive pair *votes correctly given I* if $V_{R,I}(z_d) = s(d)$. We will next show that every (d,R)-decisive pair will with high probability vote correctly given random I.

Lemma 4 *If (z_d, R) is a (d,R)-decisive pair then $s(d)\mathbf{E}_I[V_{R,I}(z_d)] \geq 1 - 18c_1c$.*

Proof: We prove the lemma for d fully θ-heavy, which we represent by the symbol a. The proof for θ-light b consists of showing that $\mathbf{E}_I[V_{R,I}(z_b)] \leq -1 + 18c_1c$, which is very similar and omitted.

Recall that

$$\widehat{f_{R,I}}(z) = \frac{1}{2^k} \sum_{p \in \{0,1\}^k} f_{R,I}(p)\chi_z(p).$$

Also, for any k-bit z, $f_{R,I}(0^k)\chi_z(0^k) = f(e_I)$. So

$$\begin{aligned} 2^k \left|\widehat{f_{R,I}}(z_a)\right| - \operatorname{sign}\left(\widehat{f_{R,I}}(z_a)\right) f(e_I) &= \operatorname{sign}\left(\widehat{f_{R,I}}(z_a)\right) \sum_{p \in \{0,1\}^k - \{0^k\}} f_{R,I}(p)\chi_{z_a}(p) \\ &= \operatorname{sign}\left(\widehat{f_{R,I}}(z_a)\right) \sum_{x \in Y} f(x \oplus e_I)\chi_a(x). \end{aligned}$$

Therefore, $V_{R,I}(z_a)$ will be 1 if and only if $\operatorname{sign}(\widehat{f_{R,I}}(z_a)) \sum_{x\in Y} f(x \oplus e_I)\chi_a(x)/(2^k - 1) \geq 2\theta/3$. Furthermore, since $0 < 1/(2^k - 1) < 2\theta/3$ for k and c as defined in Figure 4, and because also $2^k|\widehat{f_{R,I}}(z_a)| - \operatorname{sign}(\widehat{f_{R,I}}(z_a))f(e_I) \geq -1$, it follows that

$$\begin{aligned} \left|\frac{\sum_{x\in Y} f(x \oplus e_I)\chi_a(x)}{2^k - 1}\right| \geq \frac{2\theta}{3} &\Rightarrow \left|\sum_{x\in Y} f(x \oplus e_I)\chi_a(x)\right| > 1 \\ &\Rightarrow 2^k \left|\widehat{f_{R,I}}(z_a)\right| - \operatorname{sign}\left(\widehat{f_{R,I}}(z_a)\right) f(e_I) > 1 > 0 \\ &\Rightarrow \operatorname{sign}\left(\widehat{f_{R,I}}(z_a)\right) = \operatorname{sign}\left(\sum_{x\in Y} f(x \oplus e_I)\chi_a(x)\right) \end{aligned}$$

and therefore

$$\left|\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)/(2^k-1)\right| \geq 2\theta/3 \implies \text{sign}\left(\widehat{f_{R,I}}(z_a)\right)\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)/(2^k-1) \geq 2\theta/3.$$

The converse is also clearly true. Therefore,

$$V_{R,I}(z_a) = 1 \iff \left|\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)/(2^k-1)\right| \geq 2\theta/3.$$

Now for fully θ-heavy a and any R and I such that $\Delta_{R,I,a} < \theta/3$, it follows that $|\sum_{x\in Y} f(x\oplus e_I)\chi_a(x)/(2^k-1)| \geq 2\theta/3$. Also, by definition, if R is magnitude good for a then $\Pr_I[\Delta_{R,I,a} \geq \theta/3] < 9c_1c$. Therefore, for a fully θ-heavy and R magnitude good for a, the probability over random choice of I of a 1 vote is at least $1-9c_1c$ and the probability of a -1 vote at most $9c_1c$, establishing the claim that for such an a and R, $\mathbf{E}_I[V_{R,I}(z_a)] \geq 1-18c_1c$. □

Now assume that for fixed R and fixed $0<\theta<1$ and $0<\delta<1$, a multiset M of $m \geq 2\ln(2/\delta_1)/(1-18c_1c)^2$ I's is drawn uniformly at random. Then by the Hoeffding bound, any (d,R)-decisive pair will with probability at least $1-\delta_1$ vote correctly given I for a majority of the I's in M. We say in this case that the (d,R) pair *votes correctly over* M. Symbolically, if (z_d,R) is a (d,R)-decisive pair and M is a randomly selected multiset of at least m I's,

$$\Pr_M\left[\text{sign}\left[\sum_{I\in M} V_{R,I}(z_d)\right] = s(d)\right] \geq 1-\delta_1.$$

Now note that an R which is "magnitude good" is also "good" in essentially the sense used in the previous section. That is, for any R that is magnitude good for fully heavy a, with probability at least $1 - 1/c_2$ over random choice of I,

$$\text{sign}\left(\sum_{x \in Y} f(x \oplus e_I)\chi_a(x)\right) = \text{sign}\left(\hat{f}(a)\right)\chi_a(e_I).$$

Thus, applying the analysis of the previous section, it is easily seen that

$$\mathbf{E}_{I \in T}\left[\text{sign}\left(\sum_{x \in Y} f(x \oplus e_I)\chi_a(x) \cdot \sum_{x \in Y} f(x \oplus e_I \oplus e_i)\chi_a(x)\right)\right] \geq 1 - 4c_2 c.$$

So by Hoeffding, if T is a randomly chosen set of at least t I's—for t as shown in Figure 4—and if a is fully heavy, if the R chosen at line 10 is magnitude good for a, and if (R, a) votes correctly over T, then with probability at least $1 - \delta_2$ over the choice of the set T, a will be added to the candidate list L of coefficients by an execution of the statements at lines 11 through 28 of the algorithm.

Summarizing, assume that we choose a single set (call it T for consistency with the algorithm of the previous section) that contains at least $\max(m, t)$ randomly chosen I's. Then a fully heavy coefficient a will be added to the candidate list L when R is magnitude good for a and both the $V(z_a) > 0$ test at line 24 and the "decoding" of a at line 25 succeed. For T chosen as above and random R, this occurs with probability at least $(1 - 1/c_1 - 1/c_2)(1 - \delta_1 - \delta_2)$. On the other hand, a light coefficient b will be added to L only if R is not magnitude good for b or if it is magnitude good but the test $V(z_b) > 0$ incorrectly succeeds. In fact, if R is not fully magnitude good but satisfies only $\Pr_I[\Delta_{R,I,b} \geq \theta/3] < 9c_1 c$ (which is true with probability at least $1 - 1/c_1$ over choice of R), then b will be added to L with probability at most δ_1. Therefore, light b is added to L with probability at most $\delta_1 + 1/c_1$ over the choice of T and R. Note also that the test at line 26 precludes a light coefficient from being added to the list more than once for a given choice of R and T.

Therefore, in order to detect the presence of light b's in the candidate list, we will randomly choose multiple R's and T's, use each pair to add a set of coefficients to L, and then consider the sample frequency of each coefficient d appearing in L. Since R and T are chosen independently each time, for a given d each pass of the algorithm effectively produces an independent sample of a random $\{0,1\}$-valued variable having a mean value that is the probability that d is added to L over random choice of R and T. If c_1, c_2, δ_1, and δ_2 are chosen appropriately, there will be a significant gap between the L-inclusion probability for any fully heavy coefficient a and any light coefficient b. Thus once again Hoeffding can be used to choose an appropriately large sample of size ℓ that can with high probability be used to differentiate between all light coefficients in L and all of the fully heavy coefficients.

Specifically, the gap Γ between the probability of occurrence of heavy and light coefficients will be at least $1+(\delta_1+\delta_2)(1/c_1+1/c_2)-2(\delta_1+1/c_1)-(\delta_2+1/c_2)$. So we might choose the parameter λ for the Hoeffding bound to be $\Gamma/2$. We would then like to apply the Hoeffding bound to select a random sample of size ℓ sufficiently large so with probability at least $1-\delta$ the probability of occurrence for all coefficients is estimated to within λ. Since there are at most $3\ell/c\theta^2$ coefficients in L, the union bound indicates that an ℓ of at least $\ln(6\ell/c\delta\theta^2)/(2\lambda^2)$ would be sufficient.

However, there is a potential flaw in this approach having to do with the choice of λ. We do not know *a priori* which coefficients will appear in L. In particular, we will be estimating the probability of occurrence for those light coefficients that appear at least once in L, so these light coefficients will have sample mean at least $1/\ell$ even if the true mean is extremely small or even 0. In effect, the sample mean for light coefficients is biased by at most an additive $1/\ell$ factor. Notice also that for ℓ at least $\ln(6\ell/c\delta\theta^2)/(2\lambda^2)$ and given the constraints on c (see lines 1 and 2 of the algorithm), $1/\ell \leq \lambda^2/2$. Thus if we choose λ such that $\lambda^2/2+2\lambda$ is equal to Γ then, by the Hoeffding bound, removing all coefficients that occur fewer than $\ell((1-\delta_1-\delta_2)(1-1/c_1-1/c_2)-\lambda)$ times in L removes all of the light coefficients and none of the fully heavy ones, with probabilty at least $1-\delta$.

Finally, it can be verified that there are constant values c_1, c_2, δ_1, and δ_2 satisfying the given constraints (line 1. For example, we can choose $c_1 = 4$, $c_2 = 18$, $\delta_1 = 1/61$, $\delta_2 = 1/25$. With these constants fixed, $\Gamma = 7/18$, and all constraints are satisfied. Ideally, given θ and δ, the constant values (including c) should be chosen to minimize $\ell \max(m,t) n 2^k k$, which is roughly the number of operations performed by the innermost loop of the algorithm.

5 Learning DNF

In this section, we will plug the improved weak parity algorithm developed above into a version of the Harmonic Sieve, an algorithm for learning DNF with respect to the uniform distribution [15]. Before we can do this, we need to generalize the weak parity problem to a non-Boolean setting. The Sieve can then make use of the resulting generalized weak parity algorithm in order to more efficiently learn DNF.

5.1 Solving the Non-Boolean Weak Parity Problem

Earlier, we introduced the weak parity learning problem and provided several algorithms for solving it. Until now, it was assumed that the target function f was Boolean. However, the definitions of θ-heavy, weak parity learning, and weak parity algorithm immediately generalize to the case in which the target g is real-valued. We will argue here that the weak parity algorithms developed in the previous section can also be generalized to solve the weak parity problem for non-Boolean target g.

Lemma 5 *Let $g : \{0,1\}^n \to R$ be any real-valued function such that $|g| \geq 1$ and let $\theta > 0$ be any value such that there is at least one Fourier coefficient $\hat{g}(a)$ such that $|\hat{g}(a)| \geq \theta$.*

- *If Levin's algorithm is modified to use a value of $k = \lceil \log_2(1 + (2n|g|^2/\theta^2)) \rceil + 1$, it will with* *probability at least* $1/2$ *return a set containing an* n*-bit vector* a *such that* $|\hat{g}(a)| \geq \theta$*.*

- *If the improved Levin's algorithm is modified to use a value of* $k = \lceil \log_2(1 + (|g|^2/c\theta^2)) \rceil + 1$*, it will with probability at least* $1/4$ *return a set containing an* n*-bit vector* a *such that* $|\hat{g}(a)| \geq \theta$*.*

- *If Levin's algorithm with magnitude testing is modified to use a value of* $k = \lceil \log_2(1 + (|g|^2/c\theta^2)) \rceil + 1$ *then with probability at least* $1 - \delta$ *it returns a set containing all* n*-bit vectors* a *such that* $|\hat{g}(a)| \geq \theta$ *and no* n*-bit vectors* b *such that* $|\hat{g}(b)| < \theta/3$*.*

Proof: Notice first that the primary change to the earlier analysis of Levin's algorithm is that, while 1 is an upper bound on the variance σ^2 for any Boolean-valued function f, this bound does not hold for arbitrary non-Boolean g. Instead, we can use $|g|^2$ as an upper bound on the variance of g. Choosing k as given in the statement of the lemma and generalizing the earlier analysis of Levin's algorithm in the obvious ways proves first part of the lemma. Specifically, for $k = \log_2(1 + (2n|g|^2/\theta^2))\rceil$ and Y as defined in Levin's algorithm, we have that $\text{sign}(\sum_{x \in Y} g(x)\chi_a(x)) = \text{sign}(\hat{g}(a))$ with probability at least $1 - 1/2n$.

The only other significant change to the analysis involves the off-by-one error introduced by using Fourier coefficients rather than a sum over Y. In the non-Boolean setting, "extra" term $g_{R,i}(0^k)\chi_{a^T R}(0^k)$ can now have magnitude as large as $|g|$ (its magnitude was 1 for the Boolean case). However, because of the increase in k relative to the original Levin's algorithm, it will still be the case that for k as given in the statement of the lemma $|g|/|Y| = |g|/(2^k - 1) < |\hat{g}(a)|/4$, and the remainder of the off-by-one analysis for the Boolean and non-Boolean cases is identical. It can easily be verified that the off-by-one analysis for the other two algorithms generalizes in a very similar way.

The proof of Lemma 4 must also be generalized, since for non-Boolean g and any z we have that $2^k|\widehat{g_{R,I}}(z)| - \text{sign}(\widehat{g_{R,I}}(z))g(e_I) \geq -|g|$ (the bound was -1 for Boolean f). But offsetting this we can show that for the k given in the third part of the statement of the lemma, $1/(2^k-1) < 2\theta/3|g|$ (rather than $2\theta/3$ for Boolean f). Given these observations, the rest of the proof of Lemma 4 easily generalizes to non-Boolean g. □

5.2 Weak Parity Learning and DNF

The original Harmonic Sieve uses a non-Boolean version of a weak parity algorithm due to Goldreich and Levin [12] as the basis for a weak parity learning algorithm (using a construction similar to the one of Lemma 3) in a boosting-based algorithm for learning DNF. The generalized Goldreich-Levin algorithm runs in time $\tilde{O}(n|g|^6/\theta^6)$ [15] *vs.* $\tilde{O}(n|g|^2/\theta^2)$ for the final version of Levin's algorithm developed above. In the original Sieve, the weak parity algorithm is called with a membership oracle for $|g|$ bounded by $O(1/\epsilon^{2+\alpha})$ for arbitrarily small constant α, and $\theta = \Omega(1/s)$, where s is the minimum number of terms in any DNF representation of the target function. Thus in terms of the PAC parameters ϵ and s, the weak parity algorithm runs in time roughly $\tilde{O}(ns^2/\epsilon^4)$. Since the weak parity algorithm dominates the time of the inner loop of the Sieve algorithm, and the boosting loop is executed $\tilde{O}(s^2)$ times, replacing the original weak parity algorithm with the new one reduces the time bound on the Sieve from roughly $\tilde{O}(ns^8/\epsilon^12)$ to $\tilde{O}(ns^4/\epsilon^4)$.

As Klivans and Servedio have pointed out [16], replacing the original Sieve's boosting algorithm with an alternative boosting algorithm can produce further improvement. In particular, one of Freund's boosting algorithms [11] (called $\mathtt{B}_{\mathrm{Comb}}$ by Klivans and Servedio) will call the weak parity algorithm with $|g|$ bounded by $\tilde{O}(1/\epsilon)$ and θ bounded as above, yet still runs for only $\tilde{O}(s^2)$ boosting stages. This brings the time bound for the overall algorithm down to $\tilde{O}(ns^4/\epsilon^2)$, but at the expense of a somewhat more complex hypothesis than the one produced by the original Sieve. The original algorithm produces a threshold of parity functions, while the modified algorithm will produce a threshold of thresholds of parities. In fact, the top-level threshold in the hypothesis produced using $\mathtt{B}_{\mathrm{Comb}}$ may also have random variables as inputs, so this hypothesis is not necessarily even deterministic.

5.3 Sample Complexity

The final non-Boolean weak parity algorithm has sample complexity $\tilde{O}(n|g|^2/\theta^2)$. Note that in terms of its sampling of the target function g, this algorithm is *oblivious* in the sense that the algorithm makes membership queries on $MEM(g)$ without considering the target function itself: the queries are dictated by the random choices of the matrices R and sets T of n-bit vectors. Furthermore, the Harmonic Sieve simulates the membership oracle for g directly from the membership oracle for the DNF target f. That is, for any x, the only query to $MEM(f)$ needed to compute $MEM(g)(x)$ is $MEM(f)(x)$. Also, the boosting algorithm does not require that its $k_b = \tilde{O}(s^2)$ executions of the weak parity algorithm have independent probabilities of failure. Instead, it is enough to have each execution fail with probability at most δ/k_b, and then the union bound (which does not require independence) will guarantee an overall probability of success of at least $1 - \delta$.

So by drawing a single multiset $\mathcal{M}$ of R's and T's sufficient to ensure that one execution of the weak parity algorithm succeeds with probability at least $1 - \delta/k_b$, $\mathcal{M}$ plus the single set $\mathcal{Q}$ of membership queries to f dictated by $\mathcal{M}$ can be used for all executions of the weak parity algorithm, and the overall Sieve will still succeed with probability at least $1 - \delta$. Since the size of the sample required for a single execution of the weak parity algorithm depends logarithmically on its δ parameter, the size of $\mathcal{Q}$ will still be $\tilde{O}(n|g|^2/\theta^2)$. In the context of the Harmonic Sieve using $\mathtt{B}_{\mathrm{Comb}}$, this corresponds to a sample size of $\tilde{O}(ns^2/\epsilon^2)$.

5.4 Attribute Efficient Learning

As has been noted by others (see [6] and the references therein), the relevant variables of a target function can, in many learning models, be located relatively easily when a membership oracle is available. We briefly outline here an argument showing that any algorithm using membership queries to learn a projection-closed function class (i.e., a class closed under partial assignment) with respect to the uniform distribution can be made attribute efficient. Our algorithm for finding relevant variables quickly is based on an idea noted by Angluin, Hellerstein, and Karpinski [3]. The analysis in this section, when applied to the algorithm given earlier for learning DNF, implies that the factor of n in the above soft-O bounds can effectively be replaced by a factor of r.

The idea is that we will incrementally build up a set V containing relevant variables. For each candidate V we will use random sampling to attempt to verify that a projection of the target function f on V is with high probability very consistent with f over the entire n-dimensional space. Once we find a V that passes this test, we can run the learning algorithm on a simulated membership oracle for the projection of f onto V, since this projection will be a DNF expression of size no more than the size of f. On the other hand, each V that fails the test will allow us to rapidly find another relevant variable to add to V.

More specifically, to test a particular V we will first uniformly at random choose a bit-vector x of length $|V|$ and a bit-vector y of length $n - |V|$ and query f on the n-bit vector formed by assigning x to the variables in V and y to the variables in $[n] \setminus V$. Denote the value returned by the oracle by $f(x, y)$. For each such pair (x, y) we compare $f(x, y)$ with $f(x, 0)$. We repeat this for $\ell = (2/\epsilon)\ln(2n/\delta)$ random choices of (x, y)'s. If any one of these queries produces a value $f(x, y) \neq f(x, 0)$ then we have a witness that there is a relevant variable outside of V. We then flip half of the 1 bits in y, producing y', and query f on (x, y'). The result will differ from either $f(x, 0)$

or from $f(x, y)$. In either case, we will have two vectors that are half as far apart in Hamming distance as 0 and y are. Repeating this bit-flipping procedure at most $\log n$ times identifies a relevant variable that is not in V. If all tests result in equality, then with probability at least $1 - \delta/(2n)$ we have $\Pr_{x,y}[f(x, y) \neq f(x, 0)] \leq \epsilon/2$. At this point we learn $f(x, 0)$ to within an error of $\epsilon/2$ with probability at least $1 - \delta/2$.

The probability that this learning strategy fails to produce a hypothesis h that is an ϵ-approximator to f is at most, using the union bound, δ. Overall, then, this testing phase consists of at most $r \leq n$ rounds, with each round adding one relevant variable to V. Each round consists of $O((1/\epsilon)\log(n/\delta))$ queries followed by at most $\log n$ additional queries. So we can replace n with r in the earlier bounds at the expense of an additive factor of $\tilde{O}((r/\epsilon)\log^2 n)$—hiding the logarithmic factors involving $1/\delta$.

6 Lower Bounds

In this section we give some lower bounds on the sample complexity for learning classes of Boolean functions under fixed distributions. We prove the following results.

Theorem 6 *Let C be a class of Boolean functions and $0 < \epsilon < 1$ be fixed. Also, let $C_\epsilon \subseteq C$ be such that for every $f_1, f_2 \in C_\epsilon$ we have $\Pr_D[f_1 \neq f_2] \geq 2\epsilon$. Any PAC-learning algorithm with membership queries that learns C under the distribution D with accuracy ϵ and with confidence $1 - \delta$ uses at least*

$$l = \left\lfloor \log |C_\epsilon| - \log \frac{1}{1-\delta} \right\rfloor - 1$$

queries.

Proof: Let $\mathcal{A}^f_{r,s}$ be a randomized algorithm that uses a sequence of random bits r, is given a set s of m_1 random examples, and asks m_2 membership queries to f, where $m_1 + m_2 < l$. Suppose for every $f \in C_\epsilon$ we have

$$\Pr_{s,r}[D(\mathcal{A}^f_{r,s} \Delta f) \leq \epsilon] \geq 1 - \delta.$$

This implies that there is a specific sequence r_0 of bits and specific set s_0 of m_1 examples such that

$$D(\mathcal{A}^f_{r_0,s_0} \Delta f) \leq \epsilon \tag{11}$$

for at least $(1-\delta)|C_\epsilon|$ of the functions in C_ϵ. Since $\mathcal{A}_{r_0,s_0}$ asks less than l queries, each query gives a response in $\{0,1\}$, and $2^l < (1-\delta)|C_\epsilon|$, there must be two functions f_1 and f_2 in C_ϵ and satisfying (11) for which the algorithm outputs the same hypothesis $\mathcal{A}^{f_1}_{r_0,s_0} = \mathcal{A}^{f_2}_{r_0,s_0} = h$. Now

$$2\epsilon < D(f_1 \Delta f_2) \leq D(h \Delta f_1) + D(h \Delta f_2),$$

and therefore there is an i such that $D(h\Delta f_i) > \epsilon$. This is a contradiction. □

We will need the following lemma from coding theory (based on the Varshamov-Gilbert bound).

Lemma 7 *Let Σ be an alphabet with $|\Sigma| = m$ symbols. For $m > 2$ there is a code $L \subseteq \Sigma^n$ of minimum distance cn and size $|L| \geq (m^{1-c}/2)^n$.*

Proof: The lemma follows from a simple counting argument. Since each $a \in \Sigma^n$ has at most

$$1 + (m-1)\binom{n}{1} + (m-1)^2\binom{n}{2} + \cdots + (m-1)^{cn}\binom{n}{cn}$$

elements in Σ^n with distance that is less than or equal to cn, there is a code of size

$$|L| \geq \frac{m^n}{1 + (m-1)\binom{n}{1} + (m-1)^2\binom{n}{2} + \cdots + (m-1)^{cn}\binom{n}{cn}}$$

and minimum distance cn.

Now for $m = 2$ and $c < 1/2$, by [24] we have

$$\frac{2^n}{1 + \binom{n}{1} + \cdots + \binom{n}{cn}} \geq \frac{2^n}{2^{(1-d)n}} = 2^{dn}.$$

For $m > 2$ we have

$$\frac{m^n}{1 + (m-1)\binom{n}{1} + (m-1)^2\binom{n}{2} + \cdots + (m-1)^{cn}\binom{n}{cn}} \geq \frac{m^n}{m^{cn}2^n} \geq \left(\frac{m^{1-c}}{2}\right)^n.$$

□

We now show that to learn a DNF of size s on r relevant variables (and with $\epsilon < 1/4$) we need sample size nearly $s \log r$.

Theorem 8 *Any learning algorithm for the class of DNF expressions of size s on r relevant variables with respect to the uniform distribution requires sample size $\Omega(s \log(r - \log s))$.*

Proof: Let $u = r - \log s$ and $t = \log s$, and let $x_1, \ldots, x_t$ and $y_1, \ldots, y_u$ be the r variables of the DNF. For $a \in \{0,1\}^t$ we define $x^a = x_1^{a_1} \cdots x_t^{a_t}$ where $x_i^{a_i} = x_i$ if $a_i = 1$ and $x_i^{a_i} = \bar{x}_i$ if $a_i = 0$. Let $\Sigma = \{0, 1, y_1, \ldots, y_u, \bar{y}_1, \ldots, \bar{y}_u\}$. Now define the set of DNF formulas

$$C' = \left\{ \bigvee_{a \in \{0,1\}^t} x^a y_a \;\middle|\; (y_a)_{a \in \{0,1\}^t} \in \Sigma^{2^t} \right\}.$$

That is, each DNF expression in C' has 2^t terms, each containing the t x variables plus one symbol from Σ (either a y variable or a constant). Furthermore, each term in one of these expressions sets the senses (positive or negated) of the x variables differently, and all possible senses are represented. Thus the number of terms in each DNF in C' is $2^t = s$. By Lemma 7 there is $L \subset \Sigma^s$ of minimum distance cs and size

$$|L| \geq \left(\frac{(2u+2)^{1-c}}{2}\right)^s .$$

Fix such an L and define a new set C that is a subset of C':

$$C = \left\{ \bigvee_{a\in\{0,1\}^t} x^a y_a \ \left| (y_a)_{a\in\{0,1\}^t} \in L \right. \right\}.$$

For $y \in L$ we write $f_y = \bigvee_{a\in\{0,1\}^t} x^a y_a$. That is, f_y represents the DNF expression in which the ith symbol in the string y is the value of the y_a variable in the ith term of the DNF.

Now notice that for every $y^{(1)}$ and $y^{(2)}$ in L we have $f_{y^{(1)}} \oplus f_{y^{(2)}} = f_{y^{(1)}\oplus y^{(2)}}$. This follows from the fact that f_y can be also written as $\oplus_{a\in\{0,1\}^t} x^a y_a$. Now

$$\begin{aligned}
\Pr[f_{y^{(1)}} \neq f_{y^{(2)}}] &= \Pr[f_{y^{(1)}} \oplus f_{y^{(1)}} = 1] \\
&= \Pr[f_{y^{(1)}\oplus y^{(2)}} = 1] \\
&= \mathbf{E}[f_{y^{(1)}\oplus y^{(2)}}] \\
&= \frac{1}{s} \sum_{a\in\{0,1\}^t} \mathbf{E}[y_a^{(1)} \oplus y_a^{(2)}] \\
&\geq \frac{c}{2}.
\end{aligned}$$

The latter is because $y_a^{(1)}$ and $y_a^{(2)}$ are different in at least cs entries and each $y_a^{(1)} \oplus y_a^{(2)}$ that is not zero has expectation at least $1/2$.

By Theorem 6, any PAC learning algorithm using membership queries with $\epsilon = c/4$ for this class needs at least

$$\log\left(\frac{(2u+2)^{1-c}}{2}\right)^s$$

queries for any constant $c < 1$. This gives a sample complexity of $\Omega(s\log(r - \log s))$. □

7 Randomness-efficient Levin Algorithm

In [17] Kushilevitz and Mansour showed how to derandomize another algorithm for solving the same parity-finding problem solved by Levin, but they assumed that a quantity called the L_1-norm of the target function is polynomially bounded and is known. However, the L_1 norm is not polynomially bounded for the class of polynomial-size DNF expressions [20]. In this section we describe an algorithm for learning DNF with respect to uniform that is (attribute) efficient in terms of its use of random bits (its *randomness complexity*) as well as its sample complexity.

Randomization is used several times in the improved Levin's algorithm (given in Figure 3). First, the method uses the random Boolean matrix $R \in \{0,1\}^{n \times k}$ to create the pairwise independent sample. Second, it requires the set I of $t = O(\ln n)$ random vectors used for majority voting. Third, it uses randomness for Hoeffding sampling to verify that a candidate vector is θ-heavy. For the first case, which is the main focus of this section, we show that the use of small bias probability distributions [21] can reduce the number of *row* bits used by R from nk bits to essentially $k \log n$ bits. For the second case, Goldreich [13] has a method based on error-correcting codes which can decrease the *column* bits of R (which directly impacts the sample size used) from $O(n/\theta^2)$ to $O(1/\theta^2)$. We begin by describing the technique to reduce the row size of the random matrix R used in the basic algorithm (given in Figure 1) and then show how this applies to DNF learning.

We utilize the so-called biased distributions introduced by Naor and Naor [21]. A probability distribution $D : \{0,1\}^n \to [0,1]$ is called λ-*bias* if for all $a \in \{0,1\}^n \setminus \{0^n\}$ we have $|\hat{D}(a)| \leq \lambda 2^{-n}$. Naor and Naor [21] gave an explicit construction of a λ-bias probability distribution of size $O((n/\lambda)^2)$; so $O(\log(n/\lambda))$ random bits suffice for sampling from this distribution. Recall that Levin's algorithm creates the $n \times k$ random Boolean matrix R by selecting each entry randomly and independently; hence it needs kn random bits. We modify this algorithm by choosing k *independent* columns of R according to a biased distribution D over $\{0,1\}^n$. In this way, the number of random bits required is $O(k\log(n/\lambda))$, if a λ-bias D is used.

We formalize the ideas sketched above. A sequence of random variables $X_1,\ldots,X_m$ is called *pairwise δ-dependent* if for every $1\le i\ne j\le m$ and for every a,b, we have $|\Pr[X_i=a,X_j=b]-\Pr[X_i=a]\Pr[X_j=b]|\le\delta$. Next we state an observation on pairwise dependent random variables and extend Chebyshev's inequality for these types of random variables.

Claim 9 *Let $X_1,\ldots,X_m\in\{-1,+1\}$ be pairwise δ-dependent random variables. Then $|\mathbf{E}[X_iX_j]-\mathbf{E}[X_i]\mathbf{E}[X_j]|\le 4\delta$.*

Lemma 10 *Let $X_1,\ldots,X_m\in\{-1,+1\}$ be pairwise δ-dependent random variables. Suppose that for each $i\in[m]$, $\mathbf{E}[X_i]=\mu$ and $\mathbf{Var}[X_i]=\sigma^2$. Then*

$$\Pr[\mathrm{sign}(\frac{1}{m}\sum_{i=1}^m X_i)\ne\mathrm{sign}(\mu)]\le\frac{\sigma^2+4m\delta}{m\mu^2}. \tag{12}$$

Proof We start with an upper bound for the left-hand side of (12)

$$\Pr[|\frac{1}{m}\sum_{i=1}^m X_i-\mu|\ge|\mu|]\le\frac{1}{\mu^2}\mathbf{E}[(\frac{1}{m}\sum_{i=1}^m X_i-\mu)^2]\le\frac{1}{m^2\mu^2}\mathbf{E}[(\sum_i(X_i-\mu))^2].$$

Then using straightforward algebra, we get

$$\frac{1}{m^2\mu^2}[\sum_i\mathbf{E}[(X_i-\mu)^2]+\sum_{i\ne j}\mathbf{E}[(X_i-\mu)(X_j-\mu)]]=\frac{1}{m^2\mu^2}[\sum_i\mathbf{Var}[X_i]+\sum_{i\ne j}(\mathbf{E}[X_iX_j]-\mu^2)]$$

which is bounded from above by $(\sigma^2+4m\delta)/(m\mu^2)$. $\square$

Notice that by setting $\delta=1/(4m)$ and using the fact that $\sigma^2=1-\mu^2\le 1$, we obtain an upper bound of $2/(m\mu^2)$ in the lefthand side of the Chebyshev bound stated above. This is only larger by a multiplicative factor of 2 than the bound obtained from Levin's original analysis.

We will show that by choosing k independent column vectors from $\{0,1\}^n$ according to an λ-bias distribution D the sequence of random variables $X_i = Rp_i$, where $p_i \in \{0,1\}^k \setminus \{0^k\}$, is pairwise $(4\lambda/2^n)$-dependent. So, as long as $4\lambda/2^n \leq 1/(2m)$, the analysis in Subsection 4.2 still holds. In our case, choosing λ to be a fixed small constant will suffice.

Claim 11 *Let R be a n-by-k matrix with $\{0,1\}$-entries constructed by selecting k random column vectors from $\{0,1\}^n$ according to a λ-bias distribution D over $\{0,1\}^n$. Let $X_i = Rp^{(i)}$, for $p^{(i)} \in \{0,1\}^k \setminus \{0^k\}$. Then $X_1, \ldots, X_{2^k-1}$ are pairwise $(4\lambda/2^n)$-dependent random variables.*

Proof We observe first that if D is an λ-bias distribution over $\{0,1\}^n$ then $|D(x) - 2^{-n}| \leq \lambda$. Recall that since for all $a \neq 0^n$ $|\hat{D}(a)| \leq \frac{\lambda}{2^n}$ and $\hat{D}(0^n) = 2^{-n}$,

$$|D(x) - 2^{-n}| = |\sum_{a \neq 0^n} \hat{D}(a)\chi_a(x)| \leq \sum_{a \neq 0^n} |\hat{D}(a)| \leq \lambda.$$

Let p and q be any elements of $\{0,1\}^k \setminus \{0^k\}$. We need to prove that for any $p \neq q$ and any $a, b \in \{0,1\}^n$

$$|\Pr[Rp = a, Rq = b] - \Pr[Rp = a]\Pr[Rq = b]| \leq 4\lambda/2^n$$

Consider first $\Pr[Rp = a]$. Assume without loss of generality that $p_k \neq 0$. Then after choosing the first $k-1$ columns of R, the value of the last column is uniquely determine for the equation $Rp = a$ to hold, say the last column must equal to $\alpha \in \{0,1\}^n$. Hence $\Pr[Rp = a] = D(\alpha)$.

Now consider $\Pr[Rp = a, Rq = b]$, with $p \neq q \in \{0,1\}^k$ and $a, b \in \{0,1\}^n$. We assume without loss of generality that the determinant of the matrix

$$\begin{pmatrix} p_{k-1} & q_{k-1} \\ p_k & q_k \end{pmatrix}$$

is nonzero, where arithmetic is over $\mathbb{F}_2$. After choosing the first $k-2$ columns, there is a unique solution for the $(k-1)$th and kth columns, say α and β. Then $\Pr[Rp = a, Rq = b] = D(\alpha)D(\beta)$, since we draw independent columns. The difference between two quantities of the form $D(\alpha)D(\beta)$ is at most the difference between $(2^{-n} - \lambda)^2$ and $(2^{-n} + \lambda)^2$ which is at most $4\lambda/2^n$. This completes the proof. $\square$

Levin's algorithm consumes nk random bits (to choose the matrix R). For learning DNF, this costs $\tilde{O}(n \log s)$ random bits per boosting rounds and, with $\tilde{O}(s^2)$ rounds, the total number of random bits used by Harmonic Sieve is $\tilde{O}(ns^2)$. Using biased distributions to generate R, we replace the factor n by $\log n$. Furthermore, we can apply the argument sketched in Section 5.4 to replace n with r (the number of relevant variables).

Here we mention Goldreich's [13] idea of using error correcting codes for improving the modified Levin algorithm (given in Figure 3). His technique can eliminate the additional random bits required in our version in Subsection 4.3 and will achieve the same reduction in sample space (from $O(n/\theta^2)$ to $O(1/\theta^2)$).

In Goldreich's scheme, we take an *asymptotically good* binary linear (t, n, d)-code $\mathcal{C}$, i.e., $t = O(n)$ (constant rate) and $d/n = \Omega(1)$ (can tolerate a constant fraction of errors). Since $\mathcal{C}$ is a binary linear code, it has a generator matrix $G \in \{0,1\}^{t \times n}$; the encoding of a string $x \in \{0,1\}^n$ is $G \cdot x$. So the k-th bit of $G \cdot x$ is simply $\chi_{I_k}(x)$ where I_k is the subset specified by the k-th row of G. The fact that $\mathcal{C}$ can tolerate a constant fraction of errors imply that we can replace the $1/(2n)$ upper

Input: Membership oracle $MEM(f)(x, I)$ that given x and I returns $f(x \oplus e_I)$; number of input bits n; threshold $0 < \theta < 1$ such that there is at least one Fourier coefficient $\hat{f}(a)$ such that $|\hat{f}(a)| \geq \theta$
Output: With probability at least $1/2$ return set containing n-vector a such that $|\hat{f}(a)| \geq \theta$

1. Define $k \doteq \lceil \log_2(1/c_1\theta^2) \rceil + 1$, for some constant c_1
2. Let D be a λ-bias distribution over $\{0,1\}^n$, where $\lambda \doteq 1/2^{k+2}$
3. Generate k independent random vectors $\{C_1, \ldots, C_k\}$ from D
4. Define matrix $R \doteq [C_1, \ldots, C_k] \in \{0,1\}^{n \times k}$
5. Generate the set Y of n-vectors $\{R \cdot p \mid p \in \{0,1\}^k\}$
6. Let $t \doteq c_2 n$, for some constant c_2
7. Let $\mathcal{C} = \mathbb{J}(t, n, d)$ be a *asymptotically good* binary linear code tolerating an error rate of c_1. Let $G \in \{0,1\}^{t \times n}$ be the generator matrix of $\mathcal{C}$ and let *Decode* be its decoding algorithm
8. **for each** row $I \in \{0,1\}^n$ of G **do**
9.   **for each** $R \cdot p \in Y$ **do**
10.   Compute $f_{R,I}(p) \doteq f(R \cdot p \oplus e_I)$ (by calling $MEM(f)(R \cdot p, I)$)
11.   **end do**
12.   Compute (using FFT) $\widehat{f_{R,I}}$
13.   **for each** $z \in \{0,1\}^k$
14.   Compute $a_{z,I} = \mathrm{sign}\left(\widehat{f_{R,I}}(z)\right)$
15.   **end do**
16. **end do**
17. For $z \in \{0,1\}^k$ define $a_z \doteq Decode(\frac{a_{z,I_1}+1}{2}, \frac{a_{z,2}+1}{2}, \ldots, \frac{a_{z,I_t}+1}{2})$
18. **return** $\{a_z, \overline{a_z} \mid z \in \{0,1\}^k\}$

Figure 5: Derandomized Levin's algorithm.

bound on the failure probability of approximating each bit of the heavy coefficient with

$$\Pr[\mathrm{sign}(\sum_{y \in Y} f(y \oplus e_{I_k})\chi_a(y)) \neq \chi_{I_k}(a)\mathrm{sign}(\hat{f}(a))] \leq \frac{1}{c},$$

for some constant $c > 0$ and for each row I_k of G. Thus the sample size is $|Y| \geq c/\theta^2$. One can view this improvement also as a reduction in the column size of the matrix R used in Levin's algorithm, since now we can choose R of size $O(\log(n/\lambda) \times \log(1/\theta^2))$. This coding scheme is explicit and efficient since the class of Justesen codes provide such a family of asymptotically good binary linear code with an efficient decoding algorithm. We summarize the overall derandomized algorithm in Figure 5; this algorithm can also be adapted so that it is attribute efficient.

8 Future Work

While we have made progress in improving the efficiency of DNF learning, we have focused specifically on improving the subprogram for finding heavy Fourier coefficients. As noted already, Klivans and Servedio [16] have had some success at improving the efficiency of the original Harmonic Sieve algorithm for DNF learning by using a different boosting technique than the one used originally in the Sieve. However, the hypothesis produced is more complex than the one produced by the Sieve, and is not guaranteed to be deterministic. Is there a boosting algorithm that can be used to achieve the best time and sample complexity bounds given in this paper while producing hypotheses in the same function class as the original Sieve?

One goal in this research is to develop a practical DNF learning algorithm for large problems. Although the final algorithm we present is reasonably efficient asymptotically, the combination of non-trivial log factors and somewhat large constants hidden by the asymptotic analysis call into question the algorithm's practical usefulness. This raises at least two questions: can the current algorithm be tuned to perform well on large empirical problems, and if not, how can the run time for DNF learning be improved?

Another area for future study is closing the significant gap between our lower and upper bounds for the sample complexity of learning DNF.

Acknowledgments

The authors are grateful to the anonymous referee of the conference version of this paper who, among other things, reminded them that attribute efficiency is easy to achieve in the learning model studied in this paper.

References

[1] Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2(4):319-342, 1988.

[3] Dana Angluin, Lisa Hellerstein and Marek Karpinski. Learning Read-Once Formulas with Queries. *Journal of the ACM*, 40(1):185-210, 1993.

[4] Avrim Blum. Learning Boolean Functions in an Infinite Attribute Space, *Machine Learning*, 9(4):373-386, 1992.

[5] Avrim Blum, Lisa Hellerstein and Nick Littlestone. Learning in the Presence of Finitely and Infinitely Many Irrelevant Attributes. *Journal of Computer and System Sciences*, 50(1):32-40, 1995.

[6] Nader H. Bshouty and Lisa Hellerstein. Attribute-efficient Learning in Query and Mistake-bound Models. *Journal of Computer and System Sciences*, **56**(3):310-319, 1998.

[7] Ran Canetti, Guy Even and Oded Goldreich. Lower Bounds for Sampling Algorithms Estimating the Average. In *Information Processing Letters*, **53**(1):17-25, 1995.

[8] Aditi Dhagat and Lisa Hellerstein. PAC Learning with Irrelevant Attributes. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 64-74, 1994.

[9] Petr Damaschke. Adaptive versus Nonadaptive Attribute-efficient Learning. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 590-596, 1998.

[10] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. In *Proceedings of 3nd Annual Workshop on Computational Learning Theory*, 202-216, 1990.

[11] Yoav Freund. An Improved Boosting Algorithm and Its Implications on Learning Complexity. In *Proceedings of the 5th Ann. Workshop on Computational Learning Theory*, 391-398, 1992.

[12] Oded Goldreich and Leonid Levin. A Hardcore Predicate for all One-Way Functions. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, pages 25-32, 1989.

[13] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics, Volume 17, Springer-Verlag, 1999.

[14] David Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36(2), 177-222, 1988.

[15] Jeffrey C. Jackson. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. *Journal of Computer and System Sciences*, 55(3):414-440, 1997.

[16] Adam Klivans and Rocco Servedio. Boosting and Hardcore Sets. In *Proceedings of the 40th Ann. Symposium on Foundations of Computer Science*, 1999.

[17] Eyal Kushilevitz and Yishay Mansour. Learning Decision Trees using the Fourier Spectrum. *SIAM Journal on Computing*, 22(6): 1331-1348, 1993.

[18] Nick Littlestone. Learning when Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2(4):285-318, 1988.

[19] Leonid Levin. Randomness and Non-determinism. *Journal of Symbolic Logic*, 58(3):1102-1103, 1993.

[20] Yishay Mansour. An $O(n^{\log\log n})$ Learning Algorithm for DNF under the Uniform Distribution. In *Proceedings of Fifth Annual Conference on Computational Learning Theory*, pages 53–61, 1992.

[21] Joseph Naor and Moni Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, 22(4):838-856, 1993.

[22] R. Uehara, K. Tsuchida and I. Wegener. Optimal Attribute-efficient Learning of Disjunction, Parity, and Threshold Functions. In *EuroCOLT' 97, LNAI 1208 Springer*, 171-184, 1997.

[23] Leslie Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134-1142, 1984.

[24] J.H. van Lint. *Introduction to Coding Theory*. Springer-Verlag, 1982.