Journal of Machine Learning Research 15 (2014) 3815-3846 Submitted 1/13; Revised 5/14; Published 11/14

# New Results for Random Walk Learning

**Jeffrey C. Jackson** JACKSONJ@DUQ.EDU
**Karl Wimmer** WIMMERK@DUQ.EDU
*Duquesne University*
*600 Forbes Avenue*
*Pittsburgh, PA 15282-1754*

**Editor:** John Shawe-Taylor

## Abstract

In a very strong positive result for passive learning algorithms, Bshouty et al. showed that DNF expressions are efficiently learnable in the uniform random walk model. It is natural to ask whether the more expressive class of thresholds of parities (TOP) can also be learned efficiently in this model, since both DNF and TOP are efficiently uniform-learnable from queries. However, the time bounds of the algorithms of Bshouty et al. are exponential for TOP. We present a new approach to weak parity learning that leads to quasi-efficient uniform random walk learnability of TOP. We also introduce a more general random walk model and give two positive results in this new model: DNF is efficiently learnable and juntas are efficiently agnostically learnable.

**Keywords:** computational learning theory, Fourier analysis of Boolean functions, random walks, DNF learning, TOP learning

## 1. Introduction

Positive results in learning theory have often assumed that the learner has access to a membership oracle. Such a learner is *active*, actively choosing examples for which it would like information. Here we consider *passive* models where the examples are chosen randomly. A commonly studied passive model is the model where the learner has access to independently random labeled samples. In this document, we consider a model with intermediate power. We study the *random walk* model, where the learner has access to labeled samples drawn from a random walk on the Boolean cube.

Our work is another step in the line of successful Fourier-based algorithms for learning with respect to the uniform distribution. Although many of these algorithms have been active, the Low Degree Algorithm of Linial et al. (1993), the first major Fourier-based learner, was a passive learning algorithm. This algorithm uses only random examples to estimate all of the "low degree" Fourier coefficients of a target function. By showing that $AC^0$ circuits (and hence DNF expressions) are well-approximated by their Fourier coefficients of degree logarithmic in the learning parameters, Linial et al. proved that the Low Degree Algorithm can be used to learn $AC^0$ in quasi-polynomial time. Another recent passive algorithm is in a new model motivated by the smoothed analysis of algorithms initiated by Spielman and Teng (2004). In particular, Kalai et al. (2009) show how to learn DNF—and, agnostically, decision trees—with respect to so-called smoothed product distributions, essentially random perturbations of arbitrary product distributions.

When the learner is active and has access to membership queries, the learner has substantially more power. Kushilevitz and Mansour (1993) gave an algorithm for learning decision trees in polynomial time. The algorithm, based on earlier work of Goldreich and Levin (1989), uses membership queries to find the "heavy" (large magnitude) Fourier coefficients, and uses this information to construct a hypothesis. Gopalan et al. (2009) showed that this algorithm can be made to be efficient and return a hypothesis equivalent to the decision tree. Jackson (1997) gave the first polynomial-time algorithm for learning DNF formulas with membership queries. The algorithm combines the Fourier search of Kushilevitz and Mansour (1993) with the boost-by-majority algorithm of Freund (1995). Various improvements have been made to Jackson's algorithm. Recently, Kalai et al. (2009) showed how to improve this algorithm to not use boosting, but to find all heavy coefficients of one function and run an intricate procedure to construct a good hypothesis. Also, Feldman (2012) gave a simplified algorithm for this construction. In terms of agnostic learning, Gopalan et al. (2008) gave an algorithm for agnostically learning decision trees using membership queries. The possibility of agnostically learning DNF formulas with respect to the uniform distribution is left open.

A significant step forward in learning in the random walk model (and, in our opinion, a significant step in learning in any nontrivial passive model) was given by Bshouty et al. (2005). They showed that, in the random walk model, it is possible to efficiently learn the class of DNF expressions with respect to the uniform distribution. We define the oracle for the random walk model here.

**Definition 1** *The random walk oracle proceeds as follows: the first example generated by the oracle is $\langle x, f(x)\rangle$ for a uniformly random $x$ in $\{0,1\}^n$, and the initial internal state of the oracle is set to $x$. Every subsequent example is generated as follows:*

1. *Select $i \in [n]$ uniformly at random and select $b \in \{0,1\}$ uniformly at random.*
2. *Letting $x$ be the internal state, set the new internal state $x'$ to be $x'_j = x_j$ for $j \neq i$ and $x'_i = b$.*
3. *Return $\langle i, x', f(x')\rangle$.*

This transition of internal state is known as *updating* $x$. A more natural definition involves flipping rather than updating bits, where "$x'_i = b$" in the second item above is replaced with "$x'_i = 1 - x_i$". As Bshouty et al. (2005) point out, the definitions are essentially interchangeable for uniform random walks, and the updating oracle turns out to be more convenient to analyze. The accuracy of hypothesis $h$ is assessed with respect to the uniform distribution over $\{0,1\}^n$, which is the stationary distribution of this random walk. We call learning in this setting the *uniform random walk model*.

However, Bshouty et al. left the efficient learnability of polynomial-weight threshold-of-parity (TOP) functions as an open problem for the uniform random walk model; we will refer to this problem as *learning TOP*; we define the class here.

**Definition 2** *A function $f : \{0,1\}^n \to \{-1,1\}$ can be expressed as a* TOP of weight $W$ *if it can be represented as the sign of a weighted sum of parity functions, where the weights are integers, and the sum of the magnitudes of the weights is bounded by $W$. The* TOP weight *of $f$ is the minimum weight over all TOP representations of $f$.*

We will be mostly interested in the case that the weight is bounded by a polynomial in $n$. This class is equivalent to polynomial-weight polynomial threshold functions of *arbitrary* degree over $\{-1,1\}^n$. Additionally, Krause and Pudlák (1998) showed that a polynomial threshold function of weight $w$ over $\{0,1\}^n$ can be represented as a polynomial threshold function of weight $n^2w^4$ over $\{-1,1\}^n$, so this class includes polynomial-weight polynomial threshold functions over $\{0,1\}^n$ (and this latter class includes polynomial-size DNF formulas). We remark that the parity function on $n$ variables has TOP weight 1, which is minimal, while any DNF representation of this function requires $2^{n-1}$ terms, which is maximal (Lupanov, 1961). The efficient learnability of TOP is an intriguing question both because TOP is a much more expressive class than is DNF and because the membership query algorithm for efficiently learning DNF, the Harmonic Sieve (Jackson, 1997), can also efficiently learn TOP. We add that Roch (2007) showed that TOP is learnable from a modified random walk oracle that updates the bits repeatedly in some fixed order; this type of oracle is well-suited to the same analysis as Jackson's Harmonic Sieve.

Unfortunately, as Bshouty et al. point out, their approach does not seem to be capable of producing a positive TOP learning result. Actually, they give two algorithms, one using a random walk oracle and a second using a weaker oracle that can be viewed as making statistical queries to estimate noise sensitivity (which is, roughly, the probability that corrupting an input to a function changes the output). A proof due to Roch (2007) shows that time $2^{\Omega(n)}$ is required for the latter approach, and Bshouty et al.'s time bound for the former approach also becomes exponential when applied to learning even a single parity function on $\Omega(n)$ bits.

In somewhat more detail, the Bshouty et al. weak learning algorithm is loosely based on the algorithm of Goldreich and Levin (1989)—adopted for learning purposes by Kushilevitz and Mansour (1993) and therefore often referred to in learning papers as the **KM** algorithm—for locating all of the Fourier coefficients of a function whose magnitude exceeds some threshold. Bshouty et al. replace the influence-like estimates made by **KM** with noise sensitivity estimates, and they also employ a breadth-first search rather than **KM**'s depth-first approach. Each of these changes makes the modified **KM** unsuitable for finding a weak-approximating parity on $\Omega(n)$ bits: such an algorithm is essentially searching for a large-magnitude high-order Fourier coefficient, but high-order Fourier coefficients contribute only negligibly to low-order noise sensitivity estimates, which are the only estimates made by Bshouty et al.; and employing breadth-first search for a high-degree parity using the original **KM** influence-like estimates would lead to computing an exponential in $n$ number of estimates before locating the parity.

We are aware of only one approach to weakly learning parity that differs fundamentally from **KM**. This approach is based on a clever algorithm due to Levin (1993) that randomly partitions the set of $2^n$ Fourier coefficients into polynomially many bins and succeeds in finding a weak approximator if one of the bins is dominated by a single coefficient, which happens with non-negligible probability. Variants of Levin's algorithm have been proposed and analyzed by others (Bshouty et al., 2004; Feldman, 2007). But it is not at all clear how Levin's original algorithm or any of the variants could be adapted to use a random walk oracle rather than a membership oracle.

### 1.1 Our Results

Thus, it seems that a fundamentally new approach to weakly learning parity functions is needed in order to efficiently learn TOP in the random walk model. Our main result is the following:

**Theorem 3** *In the uniform random walk model, there is an algorithm that learns TOP in time polynomial in $n$ but exponential in $\log(s/\epsilon)$, where $s$ is the minimal TOP-weight of the target function and $\epsilon$ is the desired accuracy of the approximation.*

Virtually every Fourier-based learning algorithm uses the following two procedures (possibly interleaving their operation):

1. *Fourier detection:* A procedure for finding appropriate (typically heavy) Fourier coefficients of some function (not necessarily the target function).
2. *Hypothesis construction:* A procedure for constructing a hypothesis given the identified Fourier coefficients and their estimated values.

In particular, the random walk algorithm of Bshouty et al. (2005) relies on two such procedures. Specifically, it employs the hypothesis construction method of the Harmonic Sieve (Jackson, 1997) as a black box but replaces the Sieve's Fourier detection algorithm with an algorithm that implements the Bounded Sieve (Bshouty and Feldman, 2002) using a random walk oracle. Although this bounded Fourier detection approach is sufficient for learning DNF, it is inadequate for learning TOP. Specifically, the parity function on all variables (which has TOP weight 1) has no nonzero Fourier coefficients among those that the Bounded Sieve considers, and the Fourier detection phase fails.

Our algorithm also borrows its hypothesis construction step from the Harmonic Sieve, but it introduces a new Fourier detection algorithm that uses a random walk oracle to locate heavy coefficients of arbitrary degree in quasipolynomial time. Our Fourier detection algorithm borrows some underlying Fourier ideas from **KM**, but differs markedly in how it employs these ideas. A key feature is that our algorithm can be viewed as an elimination algorithm: it locates a good approximating parity function by eliminating all other possibilities. The core of our algorithm can also be viewed as an agnostic learner for parity in the random walk model; it is efficient if the optimal parity is an $O(1)$-approximator to the target. With uniform random examples alone, the best algorithms for this problem run in time $2^{O(n/\log n)}$ (Feldman et al., 2009).

Finally, we introduce a more general random walk model based on $p$-biased distributions. We briefly show why Fourier-based methods cannot efficiently learn TOP in this model. On a positive note, we generalize existing efficient learning results for DNF and juntas, showing that these classes are also efficiently learnable (juntas agnostically and properly, quite similarly to the analysis in Gopalan et al. (2008)) in the product random walk model. Roch (2007) has shown a similar result for learning DNF from random walks over the domain $[b]^n$ equipped with the uniform distribution. We also mention the work of Kalai et al. (2009) regarding the smoothed analysis model introduced by Spielman and Teng (2004). In their model, the product distribution itself is a randomly chosen perturbation of a product distribution. In the smoothed analysis model, DNF is efficiently learnable, although our random walk model and the smoothed analysis model are of incomparable strength.

We will assume that the hypothesis construction procedure of Gopalan et al. (2008) is used in our product distribution algorithm, although the subsequent improved version of their algorithm due to Feldman (2012) or even the boosting-based approach used in Jackson's Harmonic Sieve could also be used. All of these methods have comparable running times in this model. For the Fourier detection procedure, we generalize the method of Bshouty et al. (2005) to the product distribution setting, a setting they did not consider. We use an analysis similar to that of Bshouty et al. to show that the heavy low-degree Fourier coefficients of real-valued functions will, with high probability, be found by this procedure.

## 2. Preliminaries

All of our results are for versions of the well-known PAC model. In this model, the learner's goal is to recover an unknown target function $f : \{0,1\}^n \to \{-1,1\}$, where $f$ is a member of some known class of functions $\mathcal{F}$. The class $\mathcal{F}$ is typically equipped with a size measure, such as minimum TOP weight or minimum number of terms in DNF representation or minimum decision tree size. The learner is given some sort of oracle access to $f$ in the form of labeled examples $\langle x, f(x)\rangle$, and the accuracy of the hypothesis is measured with respect to a distribution $D$ (that is related to the oracle).

**Definition 4** *We say that $\mathcal{F}$ is* learnable *if there is an algorithm that, for every $\epsilon, \delta > 0$, and every target function $f \in \mathcal{F}$, given some sort of oracle access to $f$, produces a hypothesis $h : \{0,1\}^n \to \{-1,1\}$ such that*

$$\mathop{\mathbf{Pr}}_{\boldsymbol{x}\sim D}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})] \leq \epsilon.$$

*with probability $1-\delta$. The randomness is over the oracle and the algorithm. We say that such an algorithm* learns *$\mathcal{F}$.*

In this paper, we consider the case where the oracle access is a random walk oracle as described in Definition 1. In this case, the accuracy of the hypothesis is measured with respect to the stationary distribution of the random walk.

**Definition 5** *If $\mathcal{F}$ is learnable for all $\epsilon \geq 1/2 - \gamma$ for some $\gamma > 0$, we say that $\mathcal{F}$ is $\gamma$-*weakly learnable*. Further, if $\mathbf{Pr}_{\boldsymbol{x}\sim D}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})] \leq 1/2 - \gamma$, we say that $h$ is a $\gamma$-*weak approximation *to $f$.*

The run time dependence on $\delta$ for our algorithms, like virtually all PAC algorithms, is logarithmic. As is standard to simplify the exposition, we will ignore the analysis of $\delta$ parameter, and simply assume that all estimates are correct. The $\delta$ parameter is then subsumed in $O()$ notation and will henceforth be ignored.

The more interesting and challenging case is the case where there are no restrictions on the target function. In this model, our goal is to develop an algorithm that returns a function $h$ that is "competitive" with the best function from some fixed class $\mathcal{F}$; this is called agnostic learning.

**Definition 6** *We say that $\mathcal{F}$ is* agnostically learnable *if there is an algorithm that, for every $\epsilon > 0$ and* every *target function $f : \{0,1\}^n \to \{-1,1\}$, given some sort of oracle access to $f$, produces a hypothesis $h : \{0,1\}^n \to \{-1,1\}$ such that*

$$\Pr_{\boldsymbol{x}\sim D}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})] \leq \min_{g\in\mathcal{F}} \Pr_{\boldsymbol{x}\sim D}[g(\boldsymbol{x}) \neq f(\boldsymbol{x})] + \epsilon$$

*We say that such an algorithm* agnostically learns *$\mathcal{F}$.*

We will make extensive use of discrete Fourier analysis. For every vector $a \in \{0,1\}^n$, we define the function $\chi_a : \{0,1\}^n \to \{-1,1\}$ such that $\chi_a(x) = (-1)^{\sum_{i=1}^n a_i x_i} = (-1)^{a\cdot x}$. On any fixed input $x$, $\chi_a(x)$ returns the parity (1 for even parity, $-1$ for odd) of the subset of components of $x$ indexed by 1's in $a$. The set of functions $\{\chi_a\}_{a\in\{0,1\}^n}$ forms an orthonormal basis for real-valued functions over $\{0,1\}^n$, where the inner product of two functions $g$ and $h$ is taken to be $\mathbf{E}_{\boldsymbol{x}\sim\mathcal{U}}[g(\boldsymbol{x})h(\boldsymbol{x})]$. Here, $\mathcal{U}$ represents the uniform distribution on $\{0,1\}^n$. For a function $g : \{0,1\}^n \to \mathbb{R}$, we define the Fourier coefficient $\hat{g}(a) = \mathbf{E}_{\boldsymbol{x}\sim\mathcal{U}}[g(\boldsymbol{x})\chi_a(\boldsymbol{x})]$. If $g$ is Boolean (meaning that the codomain of $g$ is $\{-1,1\}$) then it is not hard to see that $\hat{g}(a) = 1 - 2\mathbf{Pr}_{\boldsymbol{x}}[\chi_a(\boldsymbol{x}) \neq g(\boldsymbol{x})]$, where again the probability is taken with respect to the uniform distribution over $\{0,1\}^n$. Thus, if $|\hat{g}(a)| \geq \gamma$, then either $\mathbf{Pr}_{\boldsymbol{x}}[\chi_a(\boldsymbol{x}) \neq g(\boldsymbol{x})] \leq 1/2 - \gamma/2$ or $\mathbf{Pr}_{\boldsymbol{x}}[-\chi_a(\boldsymbol{x}) \neq g(\boldsymbol{x})] \leq 1/2 - \gamma/2$, which implies that either $\chi_a$ or $-\chi_a$ is a $(\gamma/2)$-weak approximation to $g$.

Our primary algorithms focus on finding *heavy* Fourier coefficients—finding $a$ such that $|\hat{g}(a)| \geq \gamma$ for some threshold value $\gamma > 0$ that will be clear from context. As has just been shown, such algorithms can be viewed as $\gamma/2$-weak learning algorithms with respect to the uniform distribution. We use $\hat{\|}g\hat{\|}_2^2$ to denote $\sum_a \hat{g}^2(a)$ and $\|g\|_\infty$ to represent $\max_x |g(x)|$. Parseval's identity tells us that $\hat{\|}g\hat{\|}_2^2 = \mathbf{E}[g^2(x)]$, where the expectation is over uniform random choice of $x$; this implies that if $g$ is Boolean, $\hat{\|}g\hat{\|}_2^2 = 1$. The notation $x \oplus y$ represents the bitwise exclusive-OR of the binary vectors $x$ and $y$ (assumed to be of the same length). In later sections, instead of $\hat{f}(a)$, it will be more convenient to write $\hat{f}(A)$, where $A \subset \{1,\ldots,n\}$ is the set of coordinates at which $a$ is 1.

We will call a string in $\{0,1,*\}^n$ a *restriction* (we use such strings to represent certain subsets, which can be viewed as restrictions of larger sets). For example, $0**1*$ represents the restriction and could be viewed as representing the set of all 5-bit strings where the first bit is 0 and the fourth is 1. The *bits of a restriction* are those symbols that are not $*$'s. Note that an $n$-bit string is considered to be a restriction. For $1 \leq i \leq n$ and $b \in \{0,1,*\}$, we use the notation $\alpha + (i,b)$ to represent the restriction $\alpha'$ that is identical to $\alpha$ except that its $i$th symbol $\alpha'_i$ is $b$. We say that a restriction $a$ is *consistent* with a restriction $\alpha$ if and only if for all $i$ such that $\alpha_i \neq *$ it is the case that $a_i = \alpha_i$. A Fourier coefficient $\hat{f}(c)$ is consistent with restriction $a$ if $c$ is consistent with $a$. A sum over $a \in \alpha$ represents a sum over the set of all bit-strings $a$ consistent with $\alpha$. We use $|\alpha|$ to denote the number of non-$*$ characters in $\alpha$.

As mentioned in the introduction, the Fourier detection portion of our algorithm will be a variation of the Kushilevitz-Mansour **KM** algorithm (Kushilevitz and Mansour, 1993). The **KM** algorithm proceeds as follows. The goal is, for a given $\theta > 0$ and $g : \{0,1\}^n \to \mathbb{R}$, to find all $a \in \{0,1\}^n$ such that $|\hat{g}(a)| \geq \theta$. The number of such coefficients is clearly at most $\hat{\|g\|}_2^2/\theta^2$. The **KM** algorithm builds a binary tree with variables at each non-leaf node and restrictions at each leaf. The restriction at the leaf $v$ is consistent with the variable assignments on the path from the root to $v$. The key idea is to estimate $\sum_{a\in\alpha} \hat{g}(a)^2$. When $\alpha = *^n$, we have $\sum_{a\in\alpha} \hat{g}(a)^2 = \hat{\|g\|}_2^2$. If we find $\sum_{a\in\alpha} \hat{g}(a)^2 < \theta^2$, then we know that every desired Fourier coefficient is inconsistent with $\alpha$. If $\sum_{a\in\alpha} \hat{g}(a)^2 \geq \theta^2$ and $\alpha_i = *$ for some $i$, we can refine our partition by replacing $\alpha$ with $\alpha + (i, 0)$ and $\alpha + (i, 1)$. Continuing in this fashion, we find all the heavy Fourier coefficients. This algorithm is efficient because the number of "active" restrictions at any time is $\hat{\|g\|}_2^2/\theta^2$, so the number of leaves is at most $2\hat{\|g\|}_2^2/\theta^2$. This algorithm can efficiently learn polynomial size decision trees, but cannot efficiently learn polynomial size DNF formulas.

## 3. Finding a Heavy Parity

In this section we present and analyze our core algorithm, which given a threshold value $\theta$ and a uniform random walk oracle for a function $g : \{0,1\}^n \to \mathbb{R}$ finds the index $a$ of a Fourier coefficient such that $|\hat{g}(a)|$ (nearly) exceeds $\theta$, if such a coefficient exists. The algorithm's time bound is exponential in $\log(\|g\|_\infty/\theta)$ but is otherwise polynomial. In later sections we use this algorithm to obtain other random walk results, agnostically learning parity (in polynomial time if the accuracy $\epsilon$ is constant) and learning TOP (in polynomial time for TOPs of constant size and given constant $\epsilon$ and with significantly better run-time bound than the previous algorithms of Bshouty et al. in the general case).

### 3.1 Utility Fourier Algorithms

Our algorithm will depend on two utility algorithms that are described now.

We first require an easy lemma, which will allow us to get uniform random examples given a uniform random walk oracle:

**Lemma 7** *Let $x$ be an arbitrary initial internal state of the random walk oracle. Then with probability at least $1 - 1/t$, after $n \ln(nt)$ examples have been drawn from the oracle, every bit of $x$ will have been updated at least once.*

**Proof** The probability that the $i$th bit is not updated after $r = n\ln(nt)$ examples is $(1 - 1/n)^{n\ln(nt)} \leq \exp(-\ln(nt)) = 1/(nt)$. By the union bound, the probability that there is a bit not updated after $n\ln(nt)$ examples is at most $1/t$. ■

By making $t$ sufficiently small, we can fold the failure probability of this "reset to random" procedure into the $\delta$ parameter with only polynomial impact on the algorithm's run time. We will therefore assume in the sequel that we can, whenever needed, reset the random walk to a uniform random internal state. Among other things, this implies that an algorithm with access to a random walk oracle can efficiently draw independent random examples.

Our first utility algorithm is $\mathbf{FC}(g, a, \tau)$ (an abbreviation of **F**ourier **C**oefficient), an algorithm that takes a uniform random walk oracle for $g : \{0,1\}^n \to \mathbb{R}$, a vector $a \in \{0,1\}^n$, and $\tau > 0$ as input, and uses a uniform random set of examples to estimate the Fourier coefficient $\hat{g}(a) = \mathbf{E}_{\boldsymbol{x}}[g(\boldsymbol{x})\chi_a(\boldsymbol{x})]$ within an additive error $\tau$ of the true value. By a standard Hoeffding argument (Hoeffding, 1963), given a set of polynomial in $n$, $\|g\|_\infty$, and $1/\tau$ such examples, the mean of $g \cdot \chi_a$ over the sample provides a sufficiently accurate estimate. Therefore, **FC** can be implemented (with high probability) in time polynomial in $n$, $\|g\|_\infty$, and $1/\tau$.

Second, we will use $\mathbf{SSF}(g, \alpha, \tau)$ (an abbreviation of **S**um of **S**quared **F**ourier coefficients) to represent an algorithm that takes a random walk oracle for $g : \{0,1\}^n \to \mathbb{R}$, a restriction $\alpha$, and $\tau > 0$ as input, and returns a value $\sigma$ such that $|\sum_{a\in\alpha} \hat{g}^2(a) - \sigma| \le \tau$. If $\alpha$ contains exactly $k$ bits (or equivalently, $k$ non-$*$ entries), then it follows from the analysis of the **KM** algorithm (Kushilevitz and Mansour, 1993) that $\sum_{a\in\alpha} \hat{g}^2(a) = \mathbf{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[g(\boldsymbol{x} + \boldsymbol{y})g(\boldsymbol{x}+\boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})]$, where the expectation is over uniform choice of $\boldsymbol{x}$ from $\{0,1\}^{n-k}$ and $\boldsymbol{y}$ and $\boldsymbol{z}$ from $\{0,1\}^k$, where we use $\boldsymbol{x} + \boldsymbol{y}$ to mean the $n$-bit string formed by interleaving in the obvious way bits from $\boldsymbol{x}$ in positions where there are $*$'s in $\alpha$ with bits from $\boldsymbol{y}$ in non-$*$ positions, and where $d$ is the $k$-bit string obtained by removing all of the $*$'s from $\alpha$. Thus, **SSF** could be implemented given a membership oracle for $g$ by randomly sampling the random variable $g(\boldsymbol{x} + \boldsymbol{y})g(\boldsymbol{x} + \boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})$ and computing the sample mean. Using Hoeffding's bound, a sample of size polynomial in $1/\tau$ and $\|g\|_\infty$ suffices to ensure a sample mean within $\tau$ of the true mean, and therefore the time bound for **SSF** is polynomial in $n$, $1/\tau$, and $\|g\|_\infty$. However, when using a random walk rather than membership oracle, we have a somewhat weaker result:

**Lemma 8** *The random walk algorithm* ***SSF***$(g, \alpha, \tau)$ *can be implemented using* $\mathrm{poly}(n^{|\alpha|}, \|g\|_\infty, 1/\tau)$ *time and samples.*

**Proof** The algorithm uses examples to estimate $g(\boldsymbol{x} + \boldsymbol{y})g(\boldsymbol{x} + \boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})$. We begin by resetting the oracle to a random internal state. We then draw from the oracle until we observe a sequence of consecutive steps that collectively update all of and only the bits corresponding to non-$*$ characters in $\alpha$ (some of these bits might be updated more than once). The inputs before and after this sequence of steps give us the $\boldsymbol{x}+\boldsymbol{y}$ and $\boldsymbol{x}+\boldsymbol{z}$ values we need in order to obtain a sample from the random variable. For simplicity of analysis, we can imagine waiting for a sequence of consecutive steps that update the bits corresponding to non-$*$ characters in order, each being updated exactly once (this gives an upper bound on the time required for the actual algorithm, which will allow repeats and any ordering of updates). Then the probability that we see a sequence of $|\alpha|$ updates in order on the appropriate bits is exactly $1/n^{|\alpha|}$, and thus the running time is polynomial in $1/\tau$, $n^{|\alpha|}$, and $\|g\|_\infty$.

■

## 3.2 Intuitive Description

Given access to **SSF** and **FC**, we next describe the intuition behind our algorithm **PT** that finds a parity function weakly approximating the target, if such a function exists. If we could call **SSF** with arbitrary restrictions $\alpha$, then in order to find a weak-approximating parity function we could simply employ **SSF** as in the **KM** algorithm, which operates in levels as follows (we also assume for simplicity of exposition in this intuition section that **SSF** computes the sum of squares of coefficients exactly). At the first level, **KM** calls (a membership-oracle based version of) **SSF** with the restrictions $0*^{n-1}$ and $1*^{n-1}$. Taking $\theta$ to be the magnitude of the desired coefficient, if either of the values returned by **SSF** is less than the threshold value $\theta^2$ then we know that all coefficients with indices consistent with the corresponding restriction have magnitude less than $\theta$. For instance, if **SSF** on $0*^{n-1}$ returns a value below the threshold, we know that, if any Fourier coefficient $\hat{f}(a)$ has magnitude exceeding $\theta$, it must be the case that $a_1 = 1$. In this case, we can then continue by calling **SSF** on $10*^{n-2}$ and $11*^{n-2}$. If both returned values are above threshold, we could continue with computing the sums of squares of Fourier coefficients on the four restrictions $100*^{n-3}$, $101*^{n-3}$, $110*^{n-3}$ and $111*^{n-3}$. **KM** continues in this way, computing (estimates of) sums of squares of coefficients at level $k$ for a set of restrictions each of which contains $k$ bits. It is not hard to see that, at any level of this recursion, at most $\hat{\|g\|}_2^2/\theta^2$ restrictions will survive the threshold test. Finally, after running its **SSF** on the set of restrictions at the $n-1$ level and thresholding, **KM** can run **FC** on any surviving restrictions (now full $n$-bit vectors) to locate the desired Fourier coefficient, if it exists.

The problem with this approach in our random walk setting is that our implementation of **SSF** has a running time exponential in the level, so we cannot afford to run **SSF** at all of the levels required by **KM**. This suggests that rather than adopting the depth-oriented approach of **KM**, we might be better served by a breadth-oriented approach.

For instance, imagine that there is one especially heavy coefficient $\hat{f}(c)$ and that the sum of squares of all other coefficients is very small. Then running **SSF** on the pair of restrictions $0*^{n-1}$ and $1*^{n-1}$ would reveal the first bit of $c$ (0 if **SSF** run on the first restriction returns a large value and 1 otherwise), running the algorithm on the pair $*0*^{n-2}$ and $*1*^{n-2}$ reveals the second bit, and so on. Each of these calls to **SSF** is, in **KM** terms, at the first level, and therefore each can be run in time polynomial in $n$.

Of course, in general, the Fourier spectrum will not be so accommodating. For instance, we might be able to fix the first bit of any heavy coefficient as above, but perhaps when we call **SSF** on the pair of restrictions $*0*^{n-2}$ and $*1*^{n-2}$, both calls return values that exceed the threshold. In this case, we will further explore the coefficients consistent with one of these restrictions—let us say those consistent with $*0*^{n-2}$—by making second-level calls to **SSF**, beginning with calls on the pair of restrictions $*00*^{n-3}$ and $*01*^{n-3}$. If both of these calls return values below threshold, we know that any heavy coefficient $\hat{f}(c)$ must have $c_2 = 1$, so we can proceed with a breadthwise search at the first level. On the other hand, if exactly one of these restrictions—say $*01*^{n-3}$—returns a value above threshold, then we can continue a breadthwise search at the second level, next computing the sums of squares of coefficients consistent with $*0*0*^{n-4}$ and $*0*1*^{n-4}$.

If this breadthwise search succeeds at fixing a single bit for each of the remaining bit positions, then we will have a candidate vector $c$ on which we can run **FC**. If $c$ is rejected, or if at any point in the second-level search both of a pair of returned values are below threshold, then we will as before be able to fix $c_2 = 1$ and will continue a breadthwise search at the first level. Finally, if at any point in the second-level search we encounter a pair of above-threshold return values from **SSF**, we will begin a third-level breadthwise search over the coefficients consistent with one of these restrictions. And so on.

What we have not specified thus far is which restriction we choose to further explore when both restrictions in a pair produce above-threshold values. The answer is that we choose the restriction corresponding to the *smaller* of the two sums of squares. This is perhaps somewhat counter-intuitive; after all, we are seeking a heavy coefficient, so it might seem that our search efforts should be focused on those sets of coefficients that are most likely to contain such a coefficient. However, to a large extent our algorithm attempts to locate a heavy coefficient by *eliminating* certain sets of coefficients from further consideration. Viewed this way, choosing to focus on sets of coefficients with small sums of squares makes good sense, as these are the sets most likely to be eliminated by refined searches. What is more, we are guaranteed that every time we increase the level of our search, we are searching on a set of coefficients that has sum of squares at most one half of the sum at the previous search level. Thus, the sums decrease exponentially quickly with increases in level, which allows us to limit our calls on **SSF** to relatively shallow levels.

With this background, we are ready to formally present our algorithm.

### 3.3 PT Algorithm

**PT** and its recursive helper function **PTH**, which performs the bulk of the work, are presented as Algorithms 1 and 2. The $\alpha$ parameter of **PTH** represents the bits that have been fixed (for purposes of further exploration) by earlier search levels. The $a$ parameter is a restriction that is consistent with $\alpha$ and that further incorporates information learned to this point in the breadthwise search at the current level. In particular, $a$ is a string of bits followed by a string of $*$'s, and it essentially tells us that the index of any heavy coefficient consistent with $\alpha$ must also be consistent with (begin with the bits of) $a$.

In this paper, we use the control flow statement **throw**, which takes a value as input. As used in this algorithm, "**throw** $x$" has the effect of returning the value $x$ to the user, and the *entire* algorithm is terminated, regardless of the level of recursion at which the **throw** statement occurs. (**throw** will be used in a more general way in a later section.)

We prove the correctness of our algorithm using several lemmas.

```
Input: $\theta, \epsilon > 0$
Output: thrown value, if any, is $a \in \{0,1\}^n$ such that $|\hat{g}(a)| \geq \theta - \epsilon/2$; normal return
    guarantees that there is no $a$ such that $|\hat{g}(a)| \geq \theta + \epsilon/2$.
1: if $\theta \leq \epsilon/2$ then                                   {any coefficient will do}
2:    throw $0^n$
3: else
4:    PTH($*^n, *^n, 1, \theta, \epsilon$)
5:    return
6: end if
```

**Algorithm 1: PT**

```
Input: $\alpha \in \{0,1,*\}^n$; $a \in \{0,1,*\}^n$, where $a$ is consistent with $\alpha$; $1 \le i \le n$; $\theta > 0$;
    $0 < \epsilon < 2\theta$
Output: thrown value, if any, is $c \in \{0,1\}^n$ such that $|\hat{g}(c)| \ge \theta - \epsilon/2$; normal return
    guarantees that there is no $c$ consistent with $a$ such that $|\hat{g}(c)| \ge \theta + \epsilon/2$.
1: while $i \le n$ do
2:   $s_0 \leftarrow \mathbf{SSF}(g, \alpha + (i,0), \epsilon^2/16)$
3:   $s_1 \leftarrow \mathbf{SSF}(g, \alpha + (i,1), \epsilon^2/16)$
4:   if $s_0 < \theta^2$ and $s_1 < \theta^2$ then
5:     return
6:   else if $s_0 < \theta^2$ then
7:     $a \leftarrow a + (i,1)$
8:   else if $s_1 < \theta^2$ then
9:     $a \leftarrow a + (i,0)$
10:  else
11:    $b \leftarrow \mathrm{argmin}_b(s_b)$
12:    $\mathbf{PTH}(\alpha + (i,b), a + (i,b), i+1, \theta, \epsilon)$
13:    $a \leftarrow a + (i, 1-b)$
14:  end if
15:  $i \leftarrow i + 1$
16: end while
17: if $|\mathbf{FC}(g, a, \epsilon/2)| \ge \theta$ then
18:   throw $a$
19: else
20:   return
21: end if
```

**Algorithm 2: PTH**

**Lemma 9** *Algorithm **PT** always either throws a vector or returns normally (it does not loop infinitely).*

**Proof** Clearly, **PT** eventually terminates if **PTH** does. It is also easy to see that **PTH** terminates if called with parameter value $i$ such that $i > n$. Assume, for induction, that for some fixed $k \geq -1$, **PTH** terminates when called with $i$ such that $n - i \leq k$, and consider a call to **PTH** with $i$ such that $n - i = k + 1$. Then every recursive call to **PTH** will be made with an argument value $i$ such that $n - i \leq k$ and will therefore either return normally or throw a value, terminating **PH**. It is thus clear that, even if all recursive calls to **PTH** return normally, the while loop of **PTH** will eventually terminate, as will the call to **PTH** having $n - i = k + 1$. ■

**Lemma 10** *For any execution of **PT**, in every call to **PTH** the $\alpha$ and $i$ argument values will be such that the characters in locations $i$ through $n$ of $\alpha$ will all be $*$.*

**Proof** This is simple to verify. ■

**Lemma 11** *The restriction on **PTH** that the $a$ parameter value be consistent with the $\alpha$ value will be satisfied throughout any execution of **PT**. Furthermore, all values that $a$ takes on during any given execution of **PTH** will be consistent with the value of the $\alpha$ parameter passed to this instantiation of **PTH**.*

**Proof** The call to **PTH** from **PT** clearly satisfies the restriction. Let us then assume, for induction, that the restriction on $a$ is satisfied as long as the level of recursion never exceeds some fixed integer $k$, and consider an instance of **PTH** executing at recursion level $k$. Then since the value of $i$ in the while loop of **PTH** is never less than $i$'s initial parameter value, by Lemma 10 any bit restrictions applied to $a$ at lines 7, 9, and 13 will be consistent with $\alpha$. It follows that if **PTH** is called at line 12, initiating execution at recursion level $k + 1$, $a + (i, b)$ will be consistent with $\alpha + (i, b)$. ■

**Lemma 12** *For any execution of **PT**, in every call to **PTH** with argument values $\alpha$, $a$, $i$, $\theta$, and $\epsilon$, the depth of recursion at which the call is made will be exactly $|\alpha|$.*

**Proof** The only line for a recursive call to **PTH** is in line 12, and this is also the only line where $\alpha$ is modified. As noted in the proof of the previous lemma, the $i$ and $\alpha$ parameter values within **PTH** will always be such that position $i$ in $\alpha$ will be a $*$. Therefore, whenever this line is executed, the value of the $\alpha$ argument in the call—that is, the value of $\alpha + (i, b)$; call it $\alpha_1$—will be such that $|\alpha_1| = |\alpha| + 1$. Thus, each recursive call to **PTH** increases $|\alpha|$ by exactly 1. ■

**Lemma 13** *For any given execution of **PT**, no two distinct calls to **PTH** will have the same value for the $\alpha$ argument.*

**Proof** Suppose the claim is false and let $\alpha'$ be a restriction that is passed to **PTH** multiple times such that $|\alpha'|$ is as small as possible. By Lemma 10 every recursive call to **PTH** adds a non-$*$ bit to the $\alpha$ argument, so $\alpha'$ cannot be $*^n$. Thus, there is a maximum value—call it $j$—such that position $j$ of $\alpha'$ is not a $*$. Also by Lemma 10, the multiple calls to **PTH** with argument value $\alpha'$ must have been such that the value of their $\alpha$ parameter was $\alpha'' = \alpha' + (j, *)$. This is a contradiction to the minimality of $\alpha'$ since $|\alpha''| = |\alpha'| - 1$. ■

**Lemma 14** *For every fixed $g : \{0,1\}^n \to \mathbb{R}$ and $\theta, \epsilon > 0$, if **PT** run with these parameters throws a vector $c$, then $|\hat{g}(c)| \geq \theta - \epsilon/2$.*

**Proof** There are only two places in the algorithm that a value can be thrown. The first is at line 2 in **PT**, which is called only if $\theta \leq \epsilon/2$. In this case, $\theta - \epsilon/2 \leq 0$, so any vector $c$ will have a corresponding Fourier coefficient satisfying the requirement of the lemma. The other throw is at line 18 of **PTH**. In order for this statement to throw a vector $c$, it must be the case that $|\mathbf{FC}(g, c, \epsilon/2)| \geq \theta$, which implies $|\hat{g}(c)| \geq \theta - \epsilon/2$. ■

We now need to show that a normal return implies that all the Fourier coefficients have low magnitude. We show an equivalent statement:

**Lemma 15** *For any valid fixed $g$, $\theta$, and $\epsilon$, if there exists $c' \in \{0,1\}^n$ such that $|\hat{g}(c')| \geq \theta + \epsilon/2$, then **PT** executed on $\theta$ and $\epsilon$ and with access to a uniform random walk oracle for $g$ throws some vector $c$ such that $|\hat{g}(c)| \geq \theta - \epsilon/2$.*

**Proof**

Let us for the moment consider a function $g$ for which there is one $c'$ such that $|\hat{g}(c')| \geq \theta + \epsilon/2$ and all other coefficients $c \neq c'$ are such that $|\hat{g}(c)| < \theta - \epsilon/2$. By Lemmas 9 and 14, **PT** run on an oracle for such $g$ either throws $c'$ or returns. By Lemma 11, if **PTH** is called with a parameter $\alpha$ with which $c'$ is not consistent, **PTH** will return normally. On the other hand, if $c'$ is consistent with the $\alpha$ parameter of some call to **PTH**, then each time $s_0$ and $s_1$ are computed at lines 2 and 3 with some value assigned to $i$, $c'$ will be consistent with $\alpha + (i, c'_i)$. Thus, when we compute $s_{c'_i}$, we get a value that is at least $(\theta + \epsilon/2)^2 - \epsilon^2/16 \geq \theta^2$.

Therefore, a call to **PTH** for which $c'$ is consistent with the $\alpha$ parameter of the call will never execute the normal return at line 5. Furthermore, for such an instantiation of **PTH**, if any and all recursive calls to **PTH** at line 12 are made with $\alpha + (i, b)$ with which $c'$ is not consistent, then all will return normally. In this case, the instance of **PTH** under consideration will eventually throw $c'$ at line 18. On the other hand, if this instance makes a recursive call at line 12 using a $\alpha + (i, b)$ with which $c'$ is consistent, then it can be seen (inductively) that that call must result in $c'$ being thrown, either directly by the recursively called **PTH** itself or indirectly as a result of further recursion.

Next, consider the case in which multiple coefficients have magnitude of at least $\theta - \epsilon/2$ and at least one coefficient has magnitude at least $\theta + \epsilon/2$. Fix the index $c'$ of one such coefficient and consider a call to **PTH** with a parameter $\alpha$ with which $c'$ is consistent. The only point at which the above argument used the assumption of a single coefficient heavier than $\theta - \epsilon/2$ was in inferring that a recursive call to **PTH** at line 12 would return normally given that $c'$ was not consistent with the argument value $\alpha + (i, b)$. However, if this assumption of a normal return fails, by Lemmas 9 and 14 it must fail because the recursive call causes a vector $c$ to be thrown such that $|\hat{g}(c)| \geq \theta - \epsilon/2$.

Summarizing, we have argued that whenever **PTH** is called with a restriction $\alpha$ such that $(\theta + \epsilon/2)$-heavy $c'$ is consistent with $\alpha$, the algorithm will throw some $(\theta - \epsilon/2)$-heavy vector $c$ (which is not necessarily $c'$). Putting this together with the fact that $c'$—and every other vector—is consistent with the value $*^n$ assigned to $\alpha$ in the initial call to **PTH** gives the lemma. ■

**Theorem 16** *For any $g : \{0,1\}^n \to \mathbb{R}$, given a uniform random walk oracle for $g$, $\boldsymbol{PT}(g, \theta, \epsilon)$ runs in time polynomial in $n^{\log(\|g\|_\infty/\theta)}$ and $1/\epsilon$, throws a value $c'$ such that $|\hat{g}(c')| \geq \theta - \epsilon/2$ if there exists $c$ such that $|\hat{g}(c)| \geq \theta + \epsilon/2$, and returns normally if $|\hat{g}(c)| < \theta - \epsilon/2$ for all $c$.*

**Proof** By Lemma 15, the theorem follows by establishing the run time bound. This, in turn, depends primarily on bounding the depth of recursion of the algorithm, which we do now.

**Lemma 17** *The maximum depth of the recursion when $\boldsymbol{PT}$ is called with fixed parameter value $\theta$ and access to random walk oracle for fixed real-valued function $g$ is $O(\log(\|g\|_\infty/\theta))$.*

**Proof** Consider an arbitrary instance of **PTH** that is executing at line 11. For $b \in \{0,1\}$, define $t_b = \sum_{a \in \alpha+(i,b)} \hat{g}(a)^2$. In line 11, ideally we would like to assign the value for $b$ that minimizes $t_b/(t_b + t_{1-b})$. If we could replace the $\epsilon^2/16$ in **SSF** with 0, then the recursion at line 12 would always be called with this minimizing $b$ and we would have $t_b/(t_b + t_{1-b}) \leq 1/2$. Thus, for any call to **PTH**, it would always be the case that

$$\sum_{c \in \alpha} \hat{g}(c)^2 \leq \hat{\|g\|}_2^2 2^{-|\alpha|},$$

where we are using the fact (Lemma 12) that the recursion depth is $|\alpha|$. If the right hand side of this inequality were less than $\theta^2$, then no more recursive calls would be made. Therefore, we have that $|\alpha|$ is at most $\lceil \log_2(\hat{\|g\|}_2^2/\theta^2) \rceil$. By Parseval's, $\hat{\|g\|}_2^2 \leq L_\infty^2(g)$, which implies that the maximum depth of recursion is $O(\log(\|g\|_\infty/\theta))$ as required.

Of course, when using the actual **SSF** we can have estimation error. Since the error tolerance is $\epsilon^2/16$, if we reach line 11 then we can only assume for each $b \in \{0,1\}$ that $t_b \geq \theta^2 - \epsilon^2/16$. Recalling that $\theta \geq \epsilon/2$, we have $t_b \geq (3/4)\theta^2$. Furthermore, it could be that the value assigned to $b$ at line 11 would actually correspond to the larger of the two sums of squares; the most we can say is that $t_b \leq t_{1-b} + \epsilon^2/8$. Combining these observations yields that for the value of $b$ chosen at line 11

$$\frac{t_b}{t_b + t_{1-b}} \leq \frac{t_b}{2t_b - \epsilon^2/8} \leq \frac{t_b}{2t_b - \theta^2/2} \leq \frac{(3/4)\theta^2}{2(3/4)\theta^2 - \theta^2/2} = \frac{3}{4}$$

and it follows that

$$\sum_{a \in \alpha} \hat{g}(a)^2 \leq \hat{\|g\|}_2^2 (4/3)^{-|\alpha|}.$$

The same reading as in the error-free case gives the lemma. ■

From Lemma 13 we know that each time **PTH** is called it is passed a distinct value for the $\alpha$ argument. From Lemma 17 we also know that the maximum depth of the recursion is $O(\log(\|g\|_\infty/\theta))$, so it follows that the number of recursive calls made is at most $n^{O(\log(\|g\|_\infty/\theta))}$. Each recursive call uses at most one call to **FC**, which runs in polynomial time, and two calls to **SSF**, which run in time polynomial in $n^{\log(\|g\|_\infty/\theta)}$ and $1/\epsilon$. The theorem follows. ■

## 4. TOP Learning

The Harmonic Sieve (Jackson, 1997) learns—from a membership oracle and with accuracy measured with respect to the uniform distribution—Threshold of Parity (TOP) functions in time polynomial in their weight $w$ as well as in $n$ and $1/\epsilon$ (recall from Definition 2 that the TOP weight of a function $f$ is the minimum weight representation of that function as a threshold of an integer-weighted sum of parity functions). The algorithm's proof of correctness is based in part on the following fact (Jackson, 1997):

**Fact 18** *For every $f$ of TOP weight $w$ and every distribution $D$ over $\{0,1\}^n$, there exists a parity $\chi_a$ such that*

$$|E_D[f\chi_a]| \geq \frac{1}{2w+1}.$$

Defining $g \equiv 2^n fD$, it follows that for any TOP of weight $w$, there exists $a$ such that $|\hat{g}(a)| \geq 1/(2w+1)$. The original Sieve uses its membership queries in two ways: 1) to obtain uniform random examples for purposes of estimating hypothesis accuracy; 2) to implement **KM** in order to locate heavy $a$'s for $D$'s (and hence $g$'s) defined by a certain boosting algorithm. The original Sieve boosting algorithm (and some other boosting algorithms which could be used instead and give asymptotically better bounds) has the property that, when learning with respect to the uniform distribution, the $D$'s it defines all have the property that $2^n \max_x D(x)$ is polynomial in $1/\epsilon$. It follows that any $g$ defined using such a $D$ has $\|g\|_\infty$ that is also polynomial in $1/\epsilon$.

It is a simple matter, then, to replace the membership-query **KM** algorithm in the Sieve with the random-walk **PT** algorithm. Since $1/\theta$ is $O(w)$ (we can assume $w$ is known, since a simple binary search technique can be used otherwise), in the context of TOP learning **PT** will run in time polynomial in $n^{\log(w/\epsilon)}$. And as noted earlier, the uniform random examples required by the Sieve can be obtained using a uniform random walk oracle with $\tilde{O}(n)$ run-time cost per example. We therefore obtain the following:

**Theorem 19** *TOP is learnable in the uniform random walk model in time $n^{O(\log(w/\epsilon))}$.*

When employing certain boosting algorithms, the Harmonic Sieve produces a TOP as its hypothesis. Thus, TOP is actually *properly* learnable in the uniform random walk model in the stated time.

```
Input: $\epsilon > 0$
Output: throws $a$ such that $|\hat{f}(o)| - |\hat{f}(a)| \leq 2\epsilon$
 1: $\theta_l \leftarrow 1$
 2: try
 3:    loop
 4:       PT($\theta_l, \epsilon$)
 5:       $\theta_l \leftarrow \theta_l/2$
 6:    end loop
 7: catch $a$
 8: end try-catch
 9: if $\theta_l = 1$ or $\theta_l \leq \epsilon$ then
10:    throw $a$
11: end if
12: $\theta_u \leftarrow 2\theta_l$
13: while $\theta_u - \theta_l > \epsilon$ do
14:    try
15:       PT($(\theta_l + \theta_u)/2, \epsilon$)
16:       $\theta_u \leftarrow (\theta_l + \theta_u)/2$
17:    catch $a$
18:       $\theta_l \leftarrow (\theta_l + \theta_u)/2$
19:    end try-catch
20: end while
21: throw $a$
```

**Algorithm 3: AGPARITY**

## 5. Agnostic Parity Learning

It is straightforward to employ **PT** to agnostically learn parity in the uniform random walk model. First, some analysis. Let $o \equiv \text{argmax}_a |\hat{f}(a)|$. That is, either $\chi_o$ or $-\chi_o$ is the optimal approximating parity; let $\pm\chi_o$ represent the optimal. We will use **PT** to give a agnostic learning that is *proper*; that is, we will actually output a parity function as our hypothesis. Thus, we want an $a \in \{0,1\}^n$ such that $\mathbf{Pr}[\pm\chi_a \neq f] \leq \text{Opt} + \epsilon = \mathbf{Pr}[\pm\chi_o \neq f] + \epsilon$, where $\pm\chi_a$ represents either $\chi_a$ or $-\chi_a$, depending on which better approximates $f$. Since for any $a$, $\mathbf{Pr}[\chi_a \neq f] = (1 - \hat{f}(a))/2$, we can achieve our goal if we find an $a$ such that $|\hat{f}(o)| - |\hat{f}(a)| \leq 2\epsilon$.

The **AGPARITY** algorithm (Algorithm 3) achieves this goal, as proved in the following lemma. Note that this algorithm uses **try-catch** blocks. Any **throw** occurring within such a block is "caught" by the **catch** statement ending the block, and execution proceeds normally from that point with the thrown value assigned to the variable specified in the **catch** statement. On the other hand, if normal sequential execution encounters a **catch** statement, the **catch** is ignored and execution continues immediately after the next **end try-catch** statement. **throw** statements occurring outside a **try-catch** block behave as before, returning the specified value to the user and terminating the entire procedure.

**Lemma 20** *For any $\epsilon > 0$ and any $f : \{0,1\}^n \to \{-1,1\}$, Algorithm 3 throws a vector $a$ such that $|\hat{f}(o)| - |\hat{f}(a)| \leq 2\epsilon$. Thus,* ***AGPARITY*** *agnostically learns parity.*

**Proof** Recall (Lemma 15) that if $\mathbf{PT}(\theta_u, \epsilon)$ returns normally then there is no $a$ such that $|\hat{f}(a)| \geq \theta_u + \epsilon/2$. Thus, in the case of a normal return, $|\hat{f}(o)| < \theta_u + \epsilon/2$. On the other hand, if a call to $\mathbf{PT}(\theta_l, \epsilon)$ throws a vector $a$, then $|\hat{f}(a)| \geq \theta_l - \epsilon/2$ by Lemma 14. Therefore, if we are able to find two threshold values $0 < \theta_l < \theta_u$ such that $\theta_u - \theta_l \leq \epsilon$, $\mathbf{PT}(\theta_l, \epsilon)$ throws a vector $a$, and $\mathbf{PT}(\theta_u, \epsilon)$ returns normally, then we will have that $|\hat{f}(o)| - |\hat{f}(a)| \leq 2\epsilon$, as desired. Algorithm 3 consists for the most part of a search for such threshold values.

The algorithm begins (lines 2 through 8) by searching for a value $\theta_l$ at which $\mathbf{PT}(\theta_l, \epsilon)$ throws some vector, beginning with $\theta_l = 1$ and halving until a suitable threshold value is found. This search must terminate after $O(\log(1/\epsilon))$ iterations, since **PT** will certainly throw a vector once $\theta_l \leq \epsilon/2$. If the first call (with $\theta_l = 1$) throws a vector $a$, then **AGPARITY** can in turn also throw $a$ without any further search, since in this case $|\hat{f}(a)| \geq 1 - \epsilon/2$ and, for any Boolean $f$, $|\hat{f}(o)|$ can be at most 1. Similarly, if $\theta_l \leq \epsilon$ when a vector $a$ is first thrown, then the previous call to **PT** used a threshold value (call it $\theta_u$) that was at most $\epsilon$ greater than $\theta_l$ (since $\theta_u = 2\theta_l$), and this call returned normally. Therefore, by the analysis of the previous paragraph, $a$ can safely be thrown by **AGPARITY**.

On the other hand, if $\epsilon < \theta_l < 1$ when $a$ is thrown, then we will set $\theta_u$ to $2\theta_l$ (line 12). Throughout the remainder of the algorithm, $\theta_l$ will be the largest threshold value at which **PT** has thrown a vector, and $\theta_u$ will be the smallest value at which **PT** has returned normally. **AGPARITY** uses binary search to refine these values until they are within $\epsilon$ of one another. That is, it will call **PT** on the midpoint between the threshold values, updating $\theta_l$ if **PT** throws a vector and updating $\theta_u$ otherwise. Once the threshold values are sufficiently near one another, **AGPARITY** will throw the final vector $a$ that it has caught from **PT**. It follows immediately from the earlier analysis that this $a$ satisfies the requirements of the lemma.

■

We next analyze the run time. Clearly, the second loop of Algorithm 3, like the first, makes $O(\log(1/\epsilon))$ calls to **PT**, since the initial difference $\theta_u - \theta_l$ is less than 1 and the difference is halved each iteration. Therefore, the total number of calls to **PT** is $O(\log(1/\epsilon))$, and these calls obviously dominate the run time. Since the threshold value in each call is at least $\epsilon/4$ and $f$ is Boolean, the runtime of each call to **PT** is $O(n^{\log(1/\epsilon)})$ by Theorem 16. Therefore, we have shown:

**Theorem 21** *The class of all parity functions can be agnostically learned in the uniform random walk model in time $O(n^{\log(1/\epsilon)})$.*

## 6. Product Random Walk Model

We next turn to results for a generalization of the uniform random walk model to certain non-uniform walks. In this section, we give definitions and some preliminary observations. Subsequent sections generalize existing learning results to this model. We study product distributions, whose study in the context of learning began with Furst et al. (1991).

From this point forward, we will slightly change notation: we will index Fourier coefficients by subsets of $[n]$ rather than vectors in $\{0,1\}^n$; identifying the string $a \in \{0,1\}^n$ with the set $S = \{i | a_i = 1\}$. Further, we will take the domain of our functions over the hypercube to be $\{-1,1\}^n$ instead of $\{0,1\}^n$.

### 6.1 Properties of Product Distributions

A product distribution over $\{-1,1\}^n$ with parameter $\mu = [-1,1]^n$ is a distribution $\mathcal{D}$ where $\mathbf{E}_{\mathcal{D}}[x_i] = \mathbf{Pr}_{\mathcal{D}}[x_i = 1] - \mathbf{Pr}_{\mathcal{D}}[x_i = -1] = \mu_i$ for all $i$, and the bits $\{x_i\}_{i=1}^n$ are independent. We will denote such a distribution as $\mathcal{D}_\mu$. With respect to $\mathcal{D}_\mu$, we define the inner product of two functions $f$ and $g$ to be $\mathbf{E}_{\boldsymbol{x}\sim\mathcal{D}_\mu}[f(\boldsymbol{x})g(\boldsymbol{x})]$.

Given a vector $\mu \in [-1,1]^n$, a string $x \in \{-1,1\}^n$, and an index $i \in [n]$, define the function

$$z_i(x) = \frac{x_i - \mu_i}{\sqrt{1-\mu_i^2}}$$

and for a set $S \subseteq [n]$, define $\phi_S = \displaystyle\prod_{i\in S} z_i$. The work of Bahadur (1961) shows that the $2^n$ functions $\phi_S$ form an orthonormal basis on real valued functions on $\{-1,1\}^n$ with respect to the inner product defined in this section. We define the Fourier coefficient of $f : \{-1,1\}^n \to \mathbb{R}$ as

$$\widetilde{f}(S) = \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{D}_\mu}[f(S)\phi_S].$$

Notice that when $\mu = 0^n$, we recover the uniform distribution. Indeed, many theorems from Fourier analysis with respect to the uniform distribution are true in arbitrary product distributions, such as Parseval's identity:

$$\mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{D}_\mu}[f(\boldsymbol{x})^2] = \sum_S \widetilde{f}(S)^2.$$

We will frequently need to bound the maximum absolute value of these $\phi_S$ functions, so we will assume that the product distribution $\mathcal{D}_\mu$ is such that $\mu$ is in $[-1+c, 1-c]^n$; we refer to such a product distribution as $c$-bounded. The uniform distribution is the unique 1-bounded product distribution.

With the standard Fourier coefficients, if $\hat{f}(S)$ is heavy then the corresponding parity $\chi_S$ (or its negation) is a weak approximator to $f$. But $\phi_S$ is not a Boolean function, so it cannot necessarily be directly used as a hypothesis if $\widetilde{f}(S)$ is heavy. However, Jackson (1997) shows how to efficiently produce a weak approximating Boolean function from a heavy $\widetilde{f}(S)$ as long as $|S|$ is logarithmic and we are working in a $c$-bounded product distribution with $c$ bounded away from 0, which will be the case for our results.

We can still estimate Fourier coefficients using a modified version of **FC**. Specifically, when given $g : \{-1,1\}^n \to \mathbb{R}$ as input with tolerance parameter $\tau$, we simply estimate $\mathbf{E}[g(\boldsymbol{x})\phi_S(\boldsymbol{x})]$ to within $\pm\tau$. A simple application of a Hoeffding bound tells us that we can do this from a random walk oracle, but the number of samples required to ensure confidence in our estimate is $\mathrm{poly}(c^{-k}, 1/\tau)$ for a $c$-bounded product distribution. Thus, the factor $c^{-k}$ will often show up in our time complexity bounds. We will refer to this procedure as $\mathbf{FC}_\mu$.

## 6.2 Product Random Walk Oracle

It is not immediately clear how one would even define a random walk oracle with respect to product distributions. Fortunately for us, a very straightforward random walk has desirable properties. We use the same updating steps as before. Regardless of $\mu$, we pick a coordinate $i$ uniformly at random and replace it with a bit chosen from the one-dimensional product distribution with mean $\mu_i$. For $\mu_i = 1/2$, we replace the $i$th coordinate with a uniform random bit as in the previous sections.

It is straightforward to show that the stationary distribution of this random walk is the $\mathcal{D}_\mu$:

**Lemma 22** *The stationary distribution of the aforementioned random walk is $\mathcal{D}_\mu$, and the mixing time of this random walk is $O(n \log n)$, independent of $\mu$.*

**Proof**

It is easy to compute that with respect to $\mathcal{D}_\mu$, we have $\mathbf{Pr}_{\boldsymbol{x}\sim\mathcal{D}_\mu}[\boldsymbol{x}_i = b] = \frac{1}{2} + \frac{1}{2}b\mu_i$ for $b \in \{-1,1\}$. For a string $x \in \{-1,1\}^n$, we let $\mathcal{D}_\mu(x)$ denote the probability mass assigned to $x$. Further, let $x^{(i)}$ be $x$ with the $i$th bit flipped. Let $\boldsymbol{\alpha}$ be the current state of the random walk, and $\boldsymbol{\beta}$ the next state. It is easy to see that

$$\frac{\mathcal{D}_\mu(x^{(i)})}{\mathcal{D}_\mu(x)} = \frac{\frac{1}{2} - \frac{1}{2}x_i\mu_i}{\frac{1}{2} + \frac{1}{2}x_i\mu_i}$$

using the fact that $\mathcal{D}_\mu$ is a product distribution. Also, we have $\mathbf{Pr}[\boldsymbol{\beta} = x^{(i)}|\boldsymbol{\alpha} = x] = \frac{1}{n}(\frac{1}{2} - \frac{1}{2}x_i\mu_i)$, since the random walk step must update bit $i$ and update that bit to $-x_i$. We have a similar expression when $x^{(i)}$ and $x$ are reversed, and ultimately we have

$$\begin{aligned}
\mathcal{D}_\mu(x^{(i)})\mathbf{Pr}[\boldsymbol{\beta} = x|\boldsymbol{\alpha} = x^{(i)}] &= \mathcal{D}_\mu(x^{(i)})\frac{1}{n}\left(\frac{1}{2} + \frac{1}{2}x_i\mu_i\right) \\
&= \mathcal{D}_\mu(x)\frac{1}{n}\left(\frac{1}{2} - \frac{1}{2}x_i\mu_i\right) \\
&= \mathcal{D}_\mu(x)\mathbf{Pr}[\boldsymbol{\beta} = x^{(i)}|\boldsymbol{\alpha} = x)].
\end{aligned}$$

It follows that the chain is reversible with respect to $\mathcal{D}_\mu$, and this $\mathcal{D}_\mu$ is the stationary distribution of the Markov chain.

The mixing follows since we are updating coordinates uniformly at random, so just as in the previous sections, all $n$ coordinates are (with high probability) updated after $O(n \log n)$ steps.

■

### 6.3 The Noise Sensitivity Model

We will actually prove our product random walk results in a weaker learning model. The *product $\rho$-noise sensitivity model* is defined similarly to the $\rho$-noise sensitivity model introduced by Bshouty et al. (2005). Specifically, each call to the oracle for this model will return a 5-tuple $\langle \boldsymbol{x}, f(\boldsymbol{x}), \boldsymbol{y}, f(\boldsymbol{y}), U\rangle$, where $\boldsymbol{x}$ is generated at random from $\mathcal{D}_\mu$-biased distribution and $\boldsymbol{y}$ is constructed from $\boldsymbol{x}$ as follows: for each $i \in n$, independently and with probability $1-\rho$ update each $\boldsymbol{x}_i$ by choosing a new value from the distribution with parameter $\mu_i$ for it (if a bit of $\boldsymbol{x}$ is not updated, it is unchanged in $\boldsymbol{y}$). $U$ is the set of coordinates of the bits updated. The accuracy of the hypothesis produced will be measured with respect to $\mathcal{D}_\mu$. We refer to this model as p$\rho$NS for short. The following lemma is a generalization of Proposition 10 of Bshouty et al. (2005).

**Lemma 23** *For any fixed $0 < \rho < 1$, any algorithm in the p$\rho$NS model can be simulated in the pRW model at the cost of a $O(n \log n)$ factor in run time.*

**Proof** Our random walk can be described in the following way: given the current string $\boldsymbol{\alpha}$, the next string $\boldsymbol{\beta}$ is $\boldsymbol{\alpha}$ with probability $\mathcal{D}_\mu(\boldsymbol{\alpha}) := (1/n)\sum(\frac{1}{2} + \frac{1}{2}\boldsymbol{\alpha}_i\mu_i)$, and $\boldsymbol{\beta} = \boldsymbol{\alpha}^{(i)}$ with probability $(1/n)(\frac{1}{2} - \frac{1}{2}\boldsymbol{\alpha}_i\mu_i)$ for all $i$. Under this distribution, we can assign a bit that got updated according to $\mu_i$. If $\boldsymbol{\beta} \neq \boldsymbol{\alpha}$, then the updated bit must be the unique bit where $\boldsymbol{\beta}_i \neq \boldsymbol{\alpha}_i$. Otherwise, we randomly choose a bit that was updated (but was updated to its original value). Here, we choose $i$ with probability $(\frac{1}{2} + \frac{1}{2}\boldsymbol{\alpha}_i\mu_i)/\mathcal{D}_\mu(\boldsymbol{\alpha})$ . The total probability that $i$ is selected is $(1/n)(\frac{1}{2} - \frac{1}{2}\mu_i) + (1/n)(\frac{1}{2} + \frac{1}{2}\mu_i) = 1/n$, regardless of the value of $\boldsymbol{\alpha}$. Thus, we can treat our distribution as uniformly choosing a bit, and updating.

Fix $\rho \in (0,1)$. To simulate an oracle for *p$\rho$NS* from *pRW*, we first draw a $\mathcal{D}_\mu$ random labeled example $\langle x, f(x)\rangle$ by updating every bit, which happens after $O(n \log n)$ samples. To get $\langle y, f(y)\rangle$, we first select a random number $u$ from **Binomial**$(n, 1-\rho)$, and update precisely $u$ of the bits of $x$. We then repeatedly draw examples from the *pRW* oracle until exactly $u$ distinct bits have been updated. The resulting point is as if a random subset of $u$ bits had been updated. We take this point to be $y$, its label to be $f(y)$, and $U$ to be the set of bits updated. The resulting 5-tuple $\langle x, f(x), y, f(y), U\rangle$ is consistent with the distribution given by the *p$\rho$NS* oracle. Note that we achieve a slowdown of at most $O(n \log n)$; the worst case is when $u = n$, and we need a new example from $\mathcal{D}_\mu$. ■

## 7. Positive and Negative Results in *pRW*

Having introduced the *pRW* model, we would like to transfer our uniform random walk result for agnostically learning parity to the *pRW* model. Unfortunately, this is not possible using Fourier methods, due to the "smearing" of the Fourier spectrum of parity under product distributions where the bits are not uniformly distributed. Here, we think of parity as a function from $\{-1,1\}^n$ into $\{-1,1\}$, where $\chi_{[n]}(x) = \prod x_i$. We also define $\chi_i(x) = x_i$ for any $1 \leq i \leq n$. We will restrict ourselves to distributions where $\mu_i = \mu_j$ for all $1 \leq i \leq j \leq n$, so the only parameter is $\mu_1$.

**Claim 24** *Let $\chi_{[n]}$ be the parity function on $n$ bits. With respect to $\mathcal{D}_\mu$ where $\mu_1 = \mu_j$ for $1 \leq j \leq n$, the Fourier coefficient $\widetilde{f}(S)$ of $f = \chi_{[n]}$ is*

$$\mu_1^{n-|S|}(\sqrt{1-\mu_1^2})^{|S|}.$$

**Proof** Assume we are working under $\mathcal{D}_\mu$ where $\mu_1 = \mu_j$ for all $1 \leq j \leq n$; all expectations here are with respect to this distribution. Then

$$\begin{aligned}
\widetilde{f}(S) &= \mathbf{E}[\chi_{[n]}\phi_S] \\
&= \mathbf{E}[\prod_{i\in[n]}\chi_i \prod_{i\in S}\phi_i] \\
&= \mathbf{E}[\prod_{i\in S}\chi_i\phi_i \prod_{i\notin S}\chi_i] \\
&= \prod_{i\in S}\mathbf{E}[\chi_i\phi_i]\prod_{i\notin S}\mathbf{E}[\chi_i].
\end{aligned}$$

It is straightforward to check that $\mathbf{E}[\chi_i] = \mu_1$ and $\mathbf{E}[\chi_i\phi_i] = \frac{1}{\sqrt{1-\mu_1^2}}((1+\mu_1)(\frac{1}{2}-\frac{1}{2}\mu_1)+(1-\mu_1)(\frac{1}{2}+\frac{1}{2}\mu_1)) = \sqrt{1-\mu_1^2}$, proving the claim. ■

When $\mu_1$ is bounded away from $-1$, 0 and 1 by a constant, the expression $\sqrt{1-\mu_1^2}$ is bounded away from 0 and 1 by a constant, so every $\widetilde{f}(S)$ is exponentially small. Thus, our agnostic parity algorithm, which relies on finding heavy Fourier coefficients, cannot succeed in the product random walk model (for most values of $\mu$). Although there is a simple algorithm in the case of learning parities, this "smearing" will still happen for functions with all high-degree terms when written as a multilinear polynomial; equivalently, when all the Fourier coefficients with respect to the uniform distribution occur on vectors of high Hamming weight.

Despite this negative beginning, we are able to obtain two positive results for *pRW*. We begin by observing that many of the algorithms used in learning under the uniform distribution using Fourier analysis techniques work by estimating certain (possibly weighted) sums of squares of Fourier coefficients. Often, the algorithm efficiently estimates these sums by estimating expectations, and the correctness of these expectations depends only on the orthogonality of the $\chi_S$ functions. When only orthogonality is used, it is straightforward to extend the algorithm to product distributions. However, the structural theorems used to prove correctness may not follow, as we have just seen in the case of parity. In addition, the complexity of the algorithm may increase when extending to product distributions.

We will show two positive learning results in *pRW* (via results in $p\rho NS$): DNF formulas can be efficiently learned in the *pRW* model, and juntas can be efficiently agnostically properly learned in the *pRW* model. The efficiency of these algorithms degrades as $c$ decreases. The proofs are given in the next two sections. Both of these results rely on using Fourier detection to find heavy Fourier coefficients of low degree; such an algorithm is known as the *Bounded Sieve*.

## 8. Bounded Sieve

We begin this section with the definition of the Bounded Sieve, first appearing in Bshouty and Feldman (2002):

**Definition 25** *Let $f : \{-1, 1\}^n \to \mathbb{R}$. An algorithm with some form of oracle access to $f$ is said to perform the Bounded Sieve (with respect to $\mathcal{D}$) if, given input parameters $\theta > 0$ and $\ell \in [n]$, it outputs a list of subsets of $[n]$ such that every set $S \subseteq [n]$ satisfying $|S| \leq \ell$ and $\widetilde{f}(S)^2 \geq \theta$ appears in the list. Further, every set in the list satisfies $|S| \leq \ell$ and $\widetilde{f}(S)^2 \geq \theta/2$.*

In short, an algorithm that performs the Bounded Sieve (which we will simply refer to as the Bounded Sieve) is a Fourier detection algorithm that is only required to detect heavy low-degree Fourier coefficients. While access to the Bounded Sieve isn't enough to even learn constant-size TOP, Bshouty and Feldman (2002) showed that it is sufficient to learn polynomial-size DNF formulas with respect to the uniform distribution.

The approach taken by Bshouty and Feldman (2002) is boosting-based as is the Harmonic Sieve of Jackson (1997), so the actual implementation involves a interleaving of the Fourier detection phase (which is the weak learner) with the hypothesis construction phase (which is the boosting algorithm). Feldman (2012) gives an algorithm that runs each stage only once: it is sufficient to find all heavy low-degree Fourier coefficients of the target function; this corresponds to running the Bounded Sieve once. The contribution of Feldman (2012) is a hypothesis construction procedure that uses no extra training examples. We also note that Feldman (2012) gives a membership query algorithm for learning DNF under product distributions and claims that the algorithm also succeeds in the random walk model, although no random walk model for product distributions is clearly defined. We proceed to verify that the Bounded Sieve indeed can be performed in the product random walk model that we have defined.

**Theorem 26 (essentially Bshouty and Feldman (2002), see also Feldman (2012))** *Let $\mathcal{A}$ be an algorithm performing the Bounded Sieve (with respect to a c-bounded product distribution $\mathcal{D}_\mu$) which runs in time $\mathrm{poly}(n, \|f\|_\infty, \theta, \ell)$. Then there is a $\mathrm{poly}(n, s^{\log(2/(2-c))}, \epsilon^{-\log(2/(2-c))}, c^{-\log(2/(2$ time algorithm which learns n-variable, s-term DNF formulas to error $\epsilon$ using $\mathcal{A}$ as a black box and independent uniform random examples.*

We note that Bshouty and Feldman (2002) do not mention product distributions, but combining the work of Jackson (1997) on product distributions with their work almost immediately yields the above theorem.

Our proofs use the even weaker $p\rho NS$ oracle, As in Bshouty et al. (2005), our first goal is to estimate quantities of the form

$$\mathcal{T}(I) := \sum_{S : S \supseteq I} \rho^{|S|} \widetilde{f}(S)^2.$$

Since we only care about finding heavy Fourier coefficients $\widetilde{f}(S)$ with $|S| \leq \ell$, each such Fourier coefficient contributes at least $\rho^\ell \theta$ to $\mathcal{T}(I)$ when $S \supseteq I$. For applications to learning DNF we take $\ell$ to be $O(\log n)$ and $\theta$ to be inverse polynomial, so $\rho^{|S|}\widetilde{f}(S)^2$ is at least inverse polynomial in $n$ for the indices of Fourier coefficients that we are interested in. As an aside, the algorithm is an approximate, low-degree version of the Kushilevitz-Mansour (Kushilevitz and Mansour, 1993) algorithm applied to the "noisy" version of $f$; specifically, the function $\sum_{S \subseteq [n]} \rho^{|S|} \widetilde{f}(S) \phi_S$.

We now work towards efficiently estimating $\mathcal{T}(I)$. Given a $p\rho NS$ oracle and a set $I \subseteq [n]$, let $\mathcal{D}_\rho^{(I)}$ be the distribution of pairs $(\boldsymbol{x}, \boldsymbol{y})$ as follows: $\boldsymbol{x}$ is a random string from $\mathcal{D}_\mu$ (we will suppress the dependence on $\mu$ for clarity), and $\boldsymbol{y}$ is formed from $\boldsymbol{x}$ by updating each bit in $I$ with probability 1 and updating each bit not in $I$ with probability $1-\rho$. Using a $p\rho NS$ oracle, we can simulate this distribution. We simply keep drawing $\langle \boldsymbol{x}, f(\boldsymbol{x}), \boldsymbol{y}, f(\boldsymbol{y}), U \rangle$ until we get a 5-tuple with $I \subseteq U$. With high probability, we need at most $\text{poly}((1-\rho)^{|I|})$ examples until this happens. Then $(\boldsymbol{x}, \boldsymbol{y})$ is our desired draw from $\mathcal{D}_\rho^{(I)}$.

Define $\mathcal{T}'(I) = \mathbf{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim \mathcal{D}_\rho^{(I)}}[f(\boldsymbol{x}) f(\boldsymbol{y})]$. It is easy to see that with a $p\rho NS$ oracle we can estimate $\mathcal{T}'(I)$ (with a $\text{poly}((1-\rho)^{|I|})$ slowdown). Now we prove the analog of Claim 12 from Bshouty et al. (2005) in the product setting. Our proof will demonstrate that we only use orthonormality of the $\phi_S$ functions to achieve this result.

```
Input: $\theta > 0, \ell > 0$
Output: The returned list contains all indices of heavy or almost-heavy Fourier
    coefficients $S$ with $|S| \le \ell$.
 1: return EST$(\theta, \ell, \emptyset, \emptyset)$
```

**Algorithm 4: BOUNDED-SIEVE**

```
Input: $\theta > 0, \ell > 0, I \subseteq [n], \mathcal{L} \subseteq 2^{[n]}$
Output: The list $\mathcal{L}$ is augmented with indices of heavy or almost-heavy Fourier
    coefficients $S$ with $S \supseteq I$ and $|S| \le \ell$.
 1: Estimate $\widetilde{f}(I)^2$ to within $\pm\theta/4$.
 2: if the estimate is at least $3\theta/4$ then
 3:     $\mathcal{L} \leftarrow \mathcal{L} \cup \{I\}$
 4: end if
 5: if $|I| \ge \ell$ then
 6:     return $\mathcal{L}$
 7: end if
 8: Estimate $\mathcal{T}(I) = \sum_{S:S \supseteq I} \rho^{|S|} \widetilde{f}(S)^2$ to within $\pm\rho^\ell\theta/4$.
 9: if the estimate is at least $3\rho^\ell\theta/4$ then
10:     for all $i \in ([n] \setminus I)$ do
11:         $\mathcal{L} \leftarrow \mathcal{L} \cup$ EST$(\theta, \ell, I \cup \{i\}, \mathcal{L})$
12:     end for
13: end if
14: return $\mathcal{L}$
```

**Algorithm 5: EST**

**Claim 27** $\mathcal{T}'(I) = \sum_{S \bigcap I = \emptyset} \rho^{|S|} \widetilde{f}(S)^2$.

**Proof** All expectations in this proof are over $(\boldsymbol{x},\boldsymbol{y}) \sim \mathcal{D}_\rho^{(I)}$.

$$\begin{aligned}
\mathbf{E}[f(\boldsymbol{x})f(\boldsymbol{y})] &= \mathbf{E}[(\sum_S \widetilde{f}(S)\phi_S(\boldsymbol{x}))(\sum_T \widetilde{f}(T)\phi_T(\boldsymbol{y}))] \\
&= \sum_S \sum_T \widetilde{f}(S)\widetilde{f}(T)\mathbf{E}[\phi_S(\boldsymbol{x})\phi_T(\boldsymbol{y})] \\
&= \sum_S \sum_T \widetilde{f}(S)\widetilde{f}(T)\mathbf{E}[\prod_{i\in S} z_i(\boldsymbol{x}) \prod_{j \in T} z_j(\boldsymbol{y})],
\end{aligned}$$

where the second equality holds because the $\phi_S$ functions are orthonormal. Because we are working over product distributions, $z_i(\boldsymbol{x})$ and $z_j(\boldsymbol{x})$ are independent when $i \neq j$, and $\boldsymbol{x}$ can be replaced with $\boldsymbol{y}$ in either or both cases. Notice that if $S \setminus T$ or $T \setminus S$ is nonempty, the expectation is 0. We will assume without loss of generality that $T \setminus S$ is nonempty and $j \in T \setminus S$. Then $z_j(\boldsymbol{y})$ is independent of every other term in the expectation, and $\mathbf{E}[z_j(\boldsymbol{y})]$ is 0. So the only nonzero terms in the sum occur when $S = T$, and the sum becomes

$$\sum_S \widetilde{f}(S)^2 \mathbf{E}[\prod_{i\in S} z_i(\boldsymbol{x})z_i(\boldsymbol{y})] = \sum_S \widetilde{f}(S)^2 \prod_{i \in S} \mathbf{E}[z_i(\boldsymbol{x})z_i(\boldsymbol{y})] \tag{1}$$

using independence again.

Recall that we are trying to show $\mathcal{T}'(I) = \sum_{S \bigcap I = \emptyset} \rho^{|S|}\widetilde{f}(S)^2$, and the distribution with which we are taking expectations respect to is $\mathcal{D}_\rho^{(I)}$. To evaluate the expectation, we will consider two cases. If $i \in I$, then the $i$th bit is updated with certainty. In this case, $z_i(\boldsymbol{x})$ and $z_i(\boldsymbol{y})$ are independent and thus $\mathbf{E}[z_i(\boldsymbol{x})z_i(\boldsymbol{y})] = \mathbf{E}[z_i(\boldsymbol{x})]\mathbf{E}[z_i(\boldsymbol{y})] = 0$. For $i \notin I$, we see that $\boldsymbol{y}_i$ is updated with probability $1-\rho$. The probability distribution on $z_i(\boldsymbol{x})z_i(\boldsymbol{y})$ is as follows:

$$z_i(\boldsymbol{x})z_i(\boldsymbol{y}) = \begin{cases} \dfrac{1-\mu_i}{1+\mu_i} & \text{with probability } (\frac{1}{2}+\frac{1}{2}\mu_i)(\rho + (1-\rho)(\frac{1}{2}+\frac{1}{2}\mu_i)) \\ \dfrac{1+\mu_i}{1-\mu_i} & \text{with probability } (\frac{1}{2}-\frac{1}{2}\mu_i)(\rho + (1-\rho)(\frac{1}{2}-\frac{1}{2}\mu_i)) \\ -1 & \text{with probability } 2(\frac{1}{2}+\frac{1}{2}\mu_i)(\frac{1}{2}-\frac{1}{2}\mu_i)(1-\rho) \end{cases}$$

The first case corresponds to both bits being 1, the second for both bits being $-1$, and the third for when the bits are different. It is straightforward to verify that $\mathbf{E}[z_i(\boldsymbol{x})z_i(\boldsymbol{y})] = \rho$. So the sum in Equation 1 reduces to

$$\sum_S \widetilde{f}(S)^2 \prod_{i\in S} \mathbf{E}[z_i(\boldsymbol{x})z_i(\boldsymbol{y})] = \sum_{S:S\bigcap I = \emptyset} \rho^{|S|}\widetilde{f}(S)^2$$

as claimed. ■

Equipped with the Fourier interpretation of $\mathcal{T}'(I)$, we can now prove that $\mathcal{T}(I)$ can be efficiently estimated.

**Lemma 28** *In Step 8, $\mathcal{T}(I)$ can be efficiently estimated.*

**Proof** Define $\mathcal{T}''(I) = \sum_{S:S\bigcap I\neq\emptyset} \rho^{|S|}\widetilde{f}(S)^2$. This quantity is easy to estimate, as $\mathcal{T}''(I) + \mathcal{T}'(I) = \sum_S \rho^{|S|}\widetilde{f}(S)^2 = \mathcal{T}'(\emptyset)$. To estimate $\mathcal{T}(I) = \sum_{S:S\supseteq I} \rho^{|S|}\widetilde{f}(S)^2$ within $\gamma$, we can estimate $\mathcal{T}''(J)$ for every $J \subseteq I$ to within $\gamma/2^{|I|}$. We apply inclusion-exclusion, resulting in

$$\mathcal{T}(I) = \sum_{J\subseteq I; J\neq\emptyset} (-1)^{|J|-1}\mathcal{T}''(J).$$

There are $2^{|I|}$ many subsets, and the coefficients of $\mathcal{T}''(J)$ above are at most 1 in absolute value, so the error is at most $\gamma$. ■

Given that we can estimate $\mathcal{T}(I)$, our algorithm will perform a breadth-first search, similar to Kushilevitz and Mansour (1993). We think of the set $2^{[n]}$ as a graph in the natural way; the nodes are identified with subsets of $[n]$, and two nodes $T$ and $U$ are adjacent if the symmetric difference of $T$ and $U$ has cardinality 1. We will refer to a node $I$ as *active* when the recursive procedure is called with $I$ as the third parameter. We remark that in a breadth-first search of this graph starting at the empty set, the previously undiscovered nodes are supersets of the current node.

Starting at $I = \emptyset$, at each active node, the algorithm estimates $\widetilde{f}(I)^2$ to within $\theta/4$ and $\mathcal{T}(I)$ to within $\rho^\ell\theta/4$. The second estimate uses our procedure outlined in Lemma 28 and takes time $\text{poly}(n, (1-\rho)^\ell, 2^{|I|}, \rho^\ell\theta/2, c^{-\ell})$. The first estimate is performed via $\mathbf{FC}_\mu$ as described earlier; the required running time can be bounded by $\text{poly}(n, \|f\|_\infty, 1/\theta, c^{-|I|})$ by applying the Hoeffding bound (Hoeffding, 1963). The $c^{-|I|}$ term comes from the fact that under a $c$-bounded product distribution $\mathcal{D}_\mu$, $|\phi_I(x)| \le \left(\sqrt{\frac{2-c}{c}}\right)^{|I|} = \text{poly}(c^{-|I|})$ for every $x$. If $\mathbf{FC}_\mu$ returns that $\widetilde{f}(I)^2$ has magnitude at least $\theta/2$ then the algorithm adds $I$ to the list of $f$'s heavy Fourier coefficients. Thus if $\widetilde{f}(I)^2 \ge \theta$ then $I$ will certainly be added to the list.

The breadth-first search proceeds to the neighbors of $I$ only if $|I| < \ell$ and the estimate of $\mathcal{T}(I)$ is at least $3\rho^\ell\theta/4$. The proof is complete given two claims: first, we claim the algorithm finds all Fourier coefficients $\widetilde{f}(S)^2 \ge \theta$ and $|S| \le \ell$; and second, we claim the algorithm ends its search after visiting at most $\text{poly}(\|f\|_\infty, 1/\theta, (1-\rho)^{-\ell})$ sets $I$.

For the first claim, note that if $|S| \le \ell$ and $\widetilde{f}(S)^2 \ge \theta$, then for $I \subseteq S$, we have

$$\mathcal{T}(I) = \sum_{T:T\supseteq I} \rho^{|T|}\widetilde{f}(T)^2 \ge \rho^{|S|}\widetilde{f}(S)^2 \ge \rho^\ell\theta.$$

It follows that the breadth-first search will reach $S$. The algorithm will estimate $\widetilde{f}(S)^2$, and the set $S$ is added to the list $\mathcal{L}$.

For the second claim, we give an upper bound on the number of active nodes that the algorithm considers. First, a lemma:

**Lemma 29** *For any $f : \{-1,1\}^n \to \mathbb{R}$, $0 \le j \le n$, and $\rho \in (0,1)$, we have $\sum_{|I|=j} \mathcal{T}(I) \le \|f\|_\infty^2 \rho^j (1-\rho)^{-(j+1)}$.*

**Proof** We straightforwardly calculate:

$$\begin{aligned}
\sum_{|I|=j} \mathcal{T}(I) &= \sum_{|I|=j} \sum_{S \supseteq I} \rho^{|S|} \widetilde{f}(S)^2 \\
&= \sum_{|S| \geq j} \binom{|S|}{j} \rho^{|S|} \widetilde{f}(S)^2 \\
&\leq \left( \sum_{|S| \geq j} \widetilde{f}(S)^2 \right) \sum_{t=j}^{\infty} \binom{t}{j} \rho^t \\
&= \left( \sum_{|S| \geq j} \widetilde{f}(S)^2 \right) (1/\rho) (\frac{\rho}{1-\rho})^{j+1} \\
&\leq \|f\|_2^2 (1/\rho) (\frac{\rho}{1-\rho})^{j+1} \\
&\leq \|f\|_\infty^2 \rho^j (1-\rho)^{-(j+1)},
\end{aligned}$$

where the third equality follows from generating function identities and the fact that $\rho \in (0,1)$.

■

This implies that the number of active nodes at layer $j$ in the breadth-first search can be at most:

$$\frac{\sum_{|I|=j} \mathcal{T}(I)}{\rho^j \theta/2} = \frac{\|f\|_\infty^2 \rho^j (1-\rho)^{-(j+1)}}{\rho^j \theta/2} = 2\|f\|_\infty^2 (1/\theta)(1-\rho)^{-(j+1)}.$$

Since $j \leq \ell$, the total number of nodes the breadth-first search ever encounters is at most $(\ell+1) \cdot 2\|f\|_\infty^2 (1/\theta)(1-\rho)^{-(\ell+1)} = \mathrm{poly}(n, \|f\|_\infty, 1/\theta, (1-\rho)^{-\ell})$, as claimed.

Now that the Bounded Sieve is established, we get the following result, which is a restatement of Theorem 26:

**Theorem 30** *The class of $s$-term DNF formulas over $n$ variables can be learned with error in the $p\rho NS$ and $pRW$ models in time* $\mathrm{poly}(n, s^{\log(2/(2-c))}, \epsilon^{-\log(2/(2-c))}, c^{-\log(2/2-c)})$.

For the Fourier detection phase, we also remark that it seems difficult to transfer the running time guarantee of the **EKM** algorithm of Kalai et al. (2009). Although their paper does not specifically address membership query algorithms, their **EKM** procedure can be used as a membership query algorithm for finding heavy Fourier coefficients in product distributions with a running time independent of $c$. However, in our product random walk model, methods as rejection sampling would incur a slowdown on the order of $\mathrm{poly}(2/(2-c)^\ell)$, where $\ell$ is the size of the sets of Fourier coefficients we consider. In the case that $c$ is bounded away from 0, these extra factors are virtually constant compared to the other factors in the running time.

## 9. Agnostically Learning Juntas

In the case of agnostically learning juntas, we can use the Bounded Sieve, but we have to do slightly more work to show that this is useful. The overall complexity with our implementation of the Bounded Sieve yields a running time of $\text{poly}(n, k^k, \epsilon^{-k}, c^{-k})$ for properly agnostically learning juntas with respect to a $c$-bounded product distribution $\mathcal{D}_\mu$.

```
Input: $\epsilon, k > 0$
Output: returned function is a $k$-junta with error $\epsilon$ in computing $f$
1: Use Algorithm 4 to find $\mathcal{L}$, all Fourier coefficients $\hat{f}(S)$ such that $|S| \leq \ell = k$ and
   $\hat{f}(S)^2 \geq \theta = \epsilon^2 2^{-k}$.
2: Use $\mathbf{FC}_\mu$ to estimate $\widetilde{f}(S)$ to within $\pm\theta/4$ for each $S \in \mathcal{L}$, and call the estimate $\widehat{g}(S)$.
3: Let $g$ be the function such that $g(x) = \sum_{S \in \mathcal{S}} \widehat{g}(S)\phi_S(x)$.
4: Let $R = \{i | \sum_{S: S \ni i} |S| \widetilde{g}(S)^2 \geq \epsilon^2/k\}$, and let $g'(x) = \sum_{S \subseteq R} \widehat{g}(S)\phi_S(x)$.
5: For every $K \subseteq R$ of size $k$, let $h'_K = \text{sgn}(g'_K)$ and estimate $\text{err}_f(h'_K)$.
6: Return $h'_K$ which minimizes $\text{err}_f(h'_K)$.
```

**Algorithm 6: JUNTAS**

Our proof is very similar to the proof of Gopalan et al. (2008). In fact, our algorithm is virtually the same. However, rather than working in the model of uniform distribution plus membership queries, we extend this algorithm to product distributions as well as restricting our oracle access to a $p\rho NS$ oracle.

The algorithm of Gopalan et al. (2008) makes use of its membership queries by using KM to identify all Fourier coefficients $\hat{f}(S)$ of heavy magnitude with $|S| \leq k$. Since the size of $S$ is bounded for their purposes, it suffices to use the Bounded Sieve, just as we have already done. In showing above that DNF is efficiently learnable in the $p\rho NS$ model, we have effectively also shown that the Bounded Sieve works even in the $p\rho NS$ model. The Fourier methods used in their algorithm again only use orthogonality and are not specific to the uniform distribution. Therefore, after translating expectations to the correct product distribution, the algorithm is almost immediate.

Suppose we wish to agnostically learn a $k$-junta. We will start by running the Bounded Sieve in $p\rho NS$, stopping at level $k$ and setting the threshold $\theta = \epsilon^2 2^{-k}$. In this fashion, we find all heavy Fourier coefficients $S$ of $f$ with $|S| \leq k$. Let $\mathcal{S}$ be the set of heavy Fourier coefficients found, and set $g(x) = \sum_{S \in \mathcal{S}} \widetilde{g}(S)\phi_S(x)$. Note that we can only estimate $g$, so we use $\widehat{g}(S)$ to denote our estimate of $\widetilde{g}(S)$. Following the argument of Gopalan et al. (2008), let $R$ be the set of coordinates with high "low-degree influences"; we have $i \in R$ iff $\sum_{S \in \mathcal{S}, S \ni i} \widetilde{g}(S)^2 \leq \epsilon^2/k$. Finally, for every set $K \subseteq R$ of size $k$, estimate the error of $\text{sgn}(g'_K)$ on $f$, where $g'_K = \sum_{S \subseteq K} \widetilde{g}(S)\phi_S(x)$. The function $\text{sgn}(g'_K)$ of least error over choices of $K \subseteq R$ is our hypothesis.

We will now prove the correctness of this algorithm. We remind the reader that for a function $g : \{-1, 1\} \to \mathbb{R}$ we define $\|g\|_1$ to be $\mathbf{E}_{\boldsymbol{x}}[|g(\boldsymbol{x})|]$; in this section, the expectation is over the relevant product distribution $\mathcal{D}_\mu$. We will prove a sequence of lemmas very similar to those of Gopalan et al. (2008). First, an analog of their Lemma 13:

**Lemma 31** *Given $K \subset [n]$, and $f : \{-1,1\}^n \to \{-1,1\}$, let $f_K(x) = \sum_{S \subseteq K} \widetilde{f}(S)\phi_S(x)$. The $K$-junta that minimizes $\mathrm{err}_f(\cdot)$ is given by $h_K(x) := \mathrm{sgn}(f_K(x))$. Also, $\mathrm{err}_f(h_K) = \frac{1}{2}(1 - \|f_K\|_1)$.*

**Proof** The proof indeed follows similarly to Lemma 13 of Gopalan et al. (2008). The major difference is that in the $\{\phi_S\}$ basis, $x_i$ is not a unbiased bit. However, the $\phi_S$ functions are orthonormal, which is the property used to derive (in the $\phi$ basis) that $\mathbf{E}[\phi_S(x)|x_K = u] = \mathbf{1}[S \subseteq K]\phi_S(u)$. The rest of the argument follows nearly directly by changing $\hat{f}$ to $\widetilde{f}$. We provide the details here.

Let us fix a value $u \in \{-1,1\}^k$. As in the work of Gopalan et al. (2008), by $\boldsymbol{x}|\boldsymbol{x}_K = u$ we denote the random variable $x$ where the indices in $K$ are set according to $u$ and the rest are uniformly random. This identifies a sub-cube $C_K(u)$ of $\{-1,1\}^n$. Note that any function depending only on the coordinates in $K$ will be a constant function when restricted to $C_K(u)$. Hence, the agreement with $f$ is maximized by the function $g_K : \{-1,1\}^k \to \{-1,1\}$, where

$$g_K(u) = \mathrm{sgn}(\mathbf{E}[f(\boldsymbol{x})|\boldsymbol{x}_K = u]) = \mathrm{sgn}(\sum_{S \subseteq [n]} \widetilde{f}(S)\mathbf{E}[\phi_S(\boldsymbol{x})|\boldsymbol{x}_K = u].$$

Using the fact that we are working in a product distribution, the expected value of $\phi_{\{i\}}(\boldsymbol{x})$ is 0 if $i \notin K$. Thus,

$$\mathbf{E}[\phi_S(\boldsymbol{x})|\boldsymbol{x}_K = u] = \begin{cases} \phi_S(u) & \text{if } S \subseteq K \\ 0 & \text{otherwise} \end{cases}$$

Hence $\mathbf{E}[f(\boldsymbol{x})|\boldsymbol{x}_K = u] = \sum_{S \subseteq K} \widetilde{f}(S)\phi_S(u)$, which implies that $g_K(u) = \mathrm{sgn}(f_K(u)) = h_K(u)$. Since $\mathbf{E}[f(\boldsymbol{x})|\boldsymbol{x}_K = u] = f_K(u)$ and $f(x) \in \{-1,1\}$, we have

$$\mathbf{Pr}[f(\boldsymbol{x}) = \mathrm{sgn}(f_K(\boldsymbol{x}))|\boldsymbol{x}_K = u] = \tfrac{1}{2} + \tfrac{1}{2}(|f_K(u)|) \qquad \text{and}$$
$$\mathbf{Pr}[f(\boldsymbol{x}) \neq \mathrm{sgn}(f_K(\boldsymbol{x}))|\boldsymbol{x}_K = u] = \tfrac{1}{2} - \tfrac{1}{2}(|f_K(u)|).$$

Averaging over all $u \in \{-1,1\}^k$ with respect to the underlying product distribution and observing that the product distribution induces the appropriate product distribution on $\boldsymbol{x}_K$, we obtain $\mathrm{err}_f(h_K) = \mathbf{Pr}[h_K(x) \neq f(x)] = \frac{1}{2} - \frac{1}{2}\mathbf{E}[|f_K(\boldsymbol{x})] = \frac{1}{2}(1 - \|f_K\|_1)$. ■

Their Lemma 14 is also easily generalized:

**Lemma 32** *Let $g_K : \{-1,1\}^n \to \mathbb{R}$ be such that $\|f_K(x) - g_K(x)\|_1 < \epsilon$, and let $h'_K = \mathrm{sgn}(g_K)$. Writing $h_K = \mathrm{sgn}(f_K)$, we have $\mathrm{err}_f(h'_K) \leq \mathrm{err}_f(h_K) + 2\epsilon$.*

**Proof**

Fix $x_k = u$. Then from the previous lemma, we have

$$\mathrm{err}_f(h'_K|x_K = u) = \begin{cases} \mathrm{err}_f(h_K|x_K = u) & \text{if } \mathrm{sgn}(g_K(u)) = h_K(u) \\ \mathrm{err}_f(h_K|x_K = u) + 2|f_K(u)| & \text{if } \mathrm{sgn}(g_K(u)) \neq h_K(u) \end{cases}$$

We claim that in both cases,

$$\mathrm{err}_f(h'_K|x_K = u) \le \mathrm{err}_f(h_K|x_K = u) + 2|f_K(u) - g_K(u)|.$$

The first case is easy to see; in the second case, we use the fact that $\mathrm{sgn}(g_K) \neq h_K = \mathrm{sgn}(f_K)$, so $|f_K(u)| \le |f_K(u) - g_K(u)|$.

Averaging over all choices of $u$ with respect to the underlying product distribution, we get

$$\mathrm{err}_f(h'_K|x_K = u) \le \mathrm{err}_f(h_K) + 2\mathbf{E}[|f_K(\boldsymbol{x}) - g_K(\boldsymbol{x})|] \le \mathrm{err}_f(h_K) + 2\epsilon.$$

■

And finally, we can prove correctness of the algorithm by proving an analogue of Theorem 15 in the work of Gopalan et al. (2008):

**Theorem 33** *The described algorithm agnostically learns $k$-juntas to error* $\mathrm{Opt} + 6\epsilon$ *in the $p\rho NS$ model in time* $\mathrm{poly}(n, k^k, \epsilon^{-k}, ((2-c)/c)^k)$ *with respect to a $c$-bounded product distribution $\mathcal{D}_\mu$.*

**Proof** Let $K$ be the set such that $h_K = \mathrm{sgn}(f_K)$ has the least error Opt. After we search for heavy Fourier coefficients using the Bounded Sieve, we are guaranteed that $|\widehat{g}(S) - \widetilde{f}(S)| \le \theta$ for all $S \subseteq K$. Hence $\mathbf{E}[g_K(x) - f_K(x))^2] = \sum_{S\subseteq K}(\widetilde{f}(S)^2 - \widehat{g}(S))^2 \le 2^k\theta \le \epsilon^2$. The running time of the Bounded Sieve is certainly bounded by $\mathrm{poly}(n, k^k, \epsilon^{-k}, ((2-c)/c)^k)$.

In the next step of the algorithm, we restrict ourselves to considering variables in the set $R$, where $i \in R$ iff $\sum_{i\in S,|S|\le k} \widehat{g}(S)^2 \ge \epsilon^2/k$. Note that even if all $k$ variables from the set $K$ are dropped, the total Fourier mass involving these variables is at most $k(\epsilon^2/k) = \epsilon^2$. Hence, $\mathbf{E}[g_K(\boldsymbol{x}) - g'_K(\boldsymbol{x}))^2] \le \epsilon^2$.

Finally, we can bound $\mathbf{E}[|f_K(x) - g'_K(x)|]$ in the following manner:

$$\begin{aligned}
\mathbf{E}[|f_K(\boldsymbol{x}) - g'_K(\boldsymbol{x})|] &\le \mathbf{E}[|f_K(\boldsymbol{x}) - g_K(\boldsymbol{x})|] + \mathbf{E}[|g_K(\boldsymbol{x}) - g'_K(\boldsymbol{x})|] \\
&\le (\mathbf{E}[(f_K(\boldsymbol{x}) - g_K(\boldsymbol{x}))^2])^{1/2} + (\mathbf{E}[g_K(\boldsymbol{x}) - g'_K(\boldsymbol{x}))^2])^{1/2} \\
&\le \epsilon + \epsilon \\
&= 2\epsilon.
\end{aligned}$$

Thus by the previous lemma, $\mathrm{err}_f(g'_K) \le \mathrm{Opt} + 4\epsilon$. We estimate $\mathrm{err}_f(h'_K)$ for every $K \subseteq R$ to within $\pm\epsilon$. There must be at least one choice such that our error estimate is at most $\mathrm{Opt} + 5\epsilon$, it follows that the true error is at most $\mathrm{Opt} + 6\epsilon$ as required.

To bound the running time, we show that $|R| = O(k^2/\epsilon^2)$. Recall that we estimate each Fourier coefficient so that our estimate of $\widetilde{f}(S)^2$ (which we called $\widehat{g}(S)^2$) for $S \in \mathcal{S}$ is correct to within $\pm\theta/4$. The Bounded Sieve will only return Fourier coefficients such that $\widetilde{f}(S)^2 \ge \theta/2$, so it follows that $\widehat{g}(S)^2 \le (3\theta/4)/(\theta/2)\widetilde{f}(S)^2 = (3/2)\widetilde{f}(S)^2 \le 2\widetilde{f}(S)^2$ for each $S \in \mathcal{S}$.

Now we observe that

$$\sum_{i\in[n]}(\sum_{i\in S,|S|\le k}\widehat{\widetilde{g}}(S)^2) \le \sum_{i\in[n]}(\sum_{i\in S,|S|\le k}2\widetilde{g}(S)^2) \le \sum_{|S|\le k}2|S|\widetilde{g}(S)^2 \le 2k\sum_{S}\widetilde{g}(S)^2 \le 2k.$$

Thus, at most $2k/(\epsilon^2/k) = 2k^2/\epsilon^2$ variables can satisfy $\sum_{i\in S,|S|\le k}\widehat{\widetilde{g}}(S)^2 \ge \epsilon^2/k$. Hence $|R| \le 2k^2/\epsilon^2$. It follows that there are at most $\binom{|R|}{k} \le (2ek^2/\epsilon^2)^k$ many choices for $K$. Each estimation in Step 2 of the algorithm can be done in poly$(n, 1/\epsilon, c^{-k})$ time, so the overall running time beyond the Bounded Sieve is poly$(n, \epsilon^{-k}, k^k, c^{-k})$.

■

## 10. Further Work

Although we have made progress on learning TOP in the uniform random walk model, it would of course be preferable to have a polynomial-time algorithm. In the product model, parity of a logarithmic number of bits can be agnostically learned, and it is obvious that $n$-bit parity is learnable by a "statistical query-like" algorithm that simply observes which bits are relevant during a random walk. Can TOP be learned (quasi)-efficiently in this model? Can we remove the condition that the product distribution is $c$-bounded in the product random walk model?

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CCF-0728939. The second author thanks Ryan O'Donnell for helpful discussions. The authors thank the anonymous referees for valuable comments and suggestions which helped improve this manuscript.

## References

Raghu Raj Bahadur. A representation of the joint distribution of responses to $n$ dichotomous items. In H. Solomon, editor, *Studies in Item Analysis and Prediction*, pages 158–168. Stanford University Press, 1961.

Nader Bshouty and Vitaly Feldman. On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research*, 2:359–395, 2002.

Nader Bshouty, Jeffrey C. Jackson, and Christino Tamon. More efficient PAC learning of DNF with membership queries under the uniform distribution. *Journal of Computer and System Sciences*, 68(1):205–234, 2004.

Nader Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco Servedio. Learning DNF from Random Walks. *Journal of Computer and System Sciences*, 71(3):250–265, 2005.

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

Vitaly Feldman. Learning DNF expressions from Fourier spectrum. *Journal of Machine Learning Research - Proceedings Track*, 23:17.1–17.19, 2012.

Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On agnostic learning of parities, monomials, and halfspaces. *SIAM Journal of Computing*, 39(2):606–645, 2009.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

Merrick Furst, Jeffrey Jackson, and Sean Smith. Improved learning of $AC^0$ functions. In *Proc. 4th Annual Conference on Learning Theory (COLT)*, pages 317–325, 1991.

Oded Goldreich and Leonid Levin. A hard-core predicate for all one-way functions. In *Proc. 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 25–32, 1989.

Parikshit Gopalan, Adam Kalai, and Adam Klivans. Agnostically learning decision trees. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 527–536, 2008.

Parikshit Gopalan, Ryan O'Donnell, Rocco Servedio, Amir Shpilka, and Karl Wimmer. Testing Fourier dimensionality and sparsity. In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 500–512, 2009.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

Jeffrey C. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computing and Sys. Sci.*, 55(3):414–440, 1997.

Adam Tauman Kalai, Alex Samorodnitsky, and Shang-Hua Teng. Learning and smoothed analysis. In *Proc. 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 395–404. IEEE Computer Society, 2009. ISBN 978-0-7695-3850-1.

Matthias Krause and Pavel Pudlák. Computing boolean functions by polynomials and threshold circuits. *Computational Complexity*, 7(4):346–370, 1998.

Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, December 1993.

Leonid A. Levin. Randomness and nondeterminism. *Journal of Symbolic Logic*, 58(3): 1102–1103, 1993.

Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform and learnability. *Journal of the ACM*, 40(3):607–620, 1993.

Oleg Lupanov. Implementing the algebra of logic functions in terms of constant depth formulas in the basis &, $\vee$, $\neg$. *Dokl. Ak. Nauk. SSSR*, 136:1041–1042, 1961.

Sébastien Roch. On learning thresholds of parities and unions of rectangles in random walk models. *Random Structures and Algorithms*, 31(4):406–417, 2007.

Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.