Electronic Colloquium on Computational Complexity, Report No. 98 (2022)

Non-Adaptive Proper Learning Polynomials

Nader H. Bshouty
Dept. of Computer Science, Technion, Israel.*

July 12, 2022

Abstract

We give the first polynomial-time *non-adaptive* proper learning algorithm of Boolean sparse multivariate polynomial under the uniform distribution. Our algorithm, for s-sparse polynomial over n variables, makes $q = (s/\epsilon)^{\gamma(s,\epsilon)} \log n$ queries where $2.66 \le \gamma(s,\epsilon) \le 6.922$ and runs in $\tilde{O}(n) \cdot poly(s, 1/\epsilon)$ time. We also show that for any $\epsilon = 1/s^{O(1)}$ any non-adaptive learning algorithm must make at least $(s/\epsilon)^{\Omega(1)} \log n$ queries. Therefore, the query complexity of our algorithm is also polynomial in the optimal query complexity and optimal in n.

1 Introduction

In this paper, we study the non-adaptive learnability of the class of sparse (multivariate) polynomials over GF(2). A polynomial over GF(2) is the sum in GF(2) of monomials, where a monomial is a product of variables. It is well known that every Boolean function has a unique representation as a (multilinear) polynomial over GF(2). A Boolean function is called s-sparse polynomial if its unique polynomial expression contains at most s monomials.

In the learning model [1, 22], the learning algorithm has access to a black-box query oracle to a function f that is s-sparse polynomial. The goal is to run in $poly(n, s, 1/\epsilon)$ time, make $poly(n, s, 1/\epsilon)$ black-box queries and, with probability at least $2/3$, learn a Boolean function h that is ϵ-close to f under the uniform distribution, i.e., $\mathbf{Pr}_x[f(x) \neq h(x)] \le \epsilon$. The learning algorithm is called *proper learning* if it outputs a s-sparse polynomial. The learning algorithm is called *exact learning algorithm* if $\epsilon = 0$.

In the adaptive learning algorithms, the queries can depend on the answers to the previous queries, wherein in the non-adaptive learning algorithms, the queries are independent of the answers to the previous queries.

Adaptive proper and non-proper learning algorithms of s-sparse polynomials that run in polynomial-time and make a polynomial number of queries have been studied by many authors [2, 4, 5, 6, 9, 12, 14, 13, 16, 17, 20, 21].

Non-adaptive proper and non-proper learning algorithms of s-sparse polynomials have been studied in [15, 18, 20]. In [18]. Hellerstein and Servedio gave a non-proper learning algorithm that learns only from random examples under any distribution (PAC-learning without black-box queries, [22]) that runs in time $n^{O(n \log s)^{1/2}}$. Roth and Benedek, [15], show that for any $s \ge 2$ polynomial-time proper PAC-learning without black-box queries of s-sparse polynomials implies RP=NP. They

[*] Center for Theoretical Sciences, Guangdong Technion, (GTIIT), China.

ISSN 1433-8092

also gave a non-adaptive proper exact learning ($\epsilon = 0$) algorithm that makes $(n/\log s)^{\log s}$ black-box queries. They also show that to exactly learn s-sparse polynomial, you need at least $(n/\log s)^{\log s}$ black-box queries. See also [15, 16].

To the best of our knowledge, no polynomial-time *non-adaptive* proper (or non-proper) learning algorithm is known for s-sparse polynomials. In this paper, we give the first polynomial-time non-adaptive proper learning algorithm for sparse multivariate polynomial. We prove

Theorem 1. *There is a non-adaptive proper learning algorithm for s-sparse polynomial that runs in polynomial-time and makes $(s/\epsilon)^{O(1)} \log n$ queries.*

In [20], Roth and Benedek show that any deterministic non-adaptive learning algorithm with $\epsilon = 1/s^{O(1)}$ must make at least $(s/\epsilon)^{\Omega(1)} \log n$ queries. We prove the same bound for randomized algorithms. This shows that our query complexity in Theorem 1 is also polynomial in the optimal query complexity and optimal in n.

Our paper is organized as follows. In Section 2, we give the technique used for our algorithm in Theorem 1. In Section 3, we provide some definitions and preliminary results. The learning algorithm is given in Section 4. In Section 5, we give another algorithm that has sublinear complexity in $1/\epsilon$ when ϵ is "very" small. Then, in Section 6, we give lower and upper bounds for the query complexity of algorithms with unlimited computational power. Section 7 and the Appendix are dedicated to two proofs of two lemmas that are needed for the main algorithm.

2 Techniques

In this section, we give a brief overview of the techniques used for the main result, Theorem 1. For the other results, see Sections 5 and 6.

Our learning algorithm is composed of the following five reductions.

The first reduction reduces the non-adaptive learning of s-sparse polynomials to non-adaptive exact learning s-sparse polynomials with monomials of size at most $d = O(\log(s/\epsilon))$, i.e., degree-d s-sparse polynomials. Given a s-sparse polynomial f, we assign each variable x_i to 0 with probability[1] $p = 1 - 2^{-\Theta(\sqrt{\log s/\log(1/\epsilon)})}$. In this *partial assignment*, with high probability, monomial of size greater than d vanish. Then we learn the resulted functions. We take enough random zero assignments (partial assignments) so that, with high probability, for each small monomial M of f, there is an partial assignment q that M does not vanish under q. Collecting all the monomials of degree at most d in all the resulted functions gives a hypothesis that ϵ-approximate the target function f.

The second reduction reduces the non-adaptive exact learning of degree-d s-sparse polynomials to non-adaptive exact learning only the monomials of degree d of the degree-d s-sparse polynomial. This is done by running all the algorithms that learn the monomials of degree i, $i \in [d]$ on the target f (which is a degree-d s-sparse polynomial). After learning the monomials of degree d in f, we let g be their sum and then learn the monomials of degree $d-1$ in $f+g$ (which are the monomials of degree $d-1$ in f). Then continue the same way with $f+g$.

The third reduction reduces the non-adaptive exact learning of the monomials of degree d in the degree-d s-sparse polynomials to exact learning degree-d s-sparse *determinant-polynomials*. A

[1] We can also choose a constant p, but this value of p is the one that minimizes the query complexity of the tester.

degree-d s-sparse determinant-polynomials is a polynomial over the new nd variables $\{y_{i,j}\}_{i\in[n],j\in[d]}$ of the form

$$\sum_{I\in S}\det(\mathcal{Y}^{(I)})$$

where S is a set of d-subsets of $[n]$, $|S|=s$ and for $I=\{i_1,\ldots,i_d\}\in S$, $\mathcal{Y}^{(I)}$ is the $d\times d$ matrix where $\mathcal{Y}^{(I)}_{k,j}=y_{i_k,j}$. Notice that the degree-d s-sparse determinant-polynomials is a homogeneous degree-d $(d!s)$-sparse polynomial over dn variables. For this reduction, we use the operator ϕ_d defined in [11] that changes the degree-d s-sparse polynomial f to degree-d s-sparse determinant-polynomial. This operator is linear, changes each monomial $\Pi_{i\in I}x_i$ of degree d in f to $\det(\mathcal{Y}^{(I)})$, removes monomials of degree less than d, and each black-box query to $\phi_d f$ can be simulated by $2^d=poly(s/\epsilon)$ queries to f. Obviously, if we can learn S in the degree-d s-sparse determinant-polynomial $\phi_d f$, we can learn the monomials of degree d of the degree-d s-sparse polynomials f.

Notice that the monomials of degree-d s-sparse determinant-polynomials are of the form $y_{i_1,1}y_{i_2,2}\cdots y_{i_d,d}$. This reduction aims to change the polynomial to a homogeneous polynomial that their monomials are of the form $y_{i_1,1}\cdots y_{i_d,d}$, so we can apply the following (fourth) reduction, which can be applied only to such polynomials.

The fourth reduction uses the simulation in Lemma 1 (see also [11]), which shows that any black-box query in $\mathrm{GF}(2^t)^n$ to a homogeneous degree-d polynomial f with monomials of the form $y_{i_1,1}\cdots y_{i_d,d}$ can be simulated in $poly(2^d,t)\tilde{O}(n)=poly(s/\epsilon)\tilde{O}(n)$ time by $O(2^{1.66d}t)=poly(s/\epsilon)t$ black-box queries in $\mathrm{GF}(2)^n$. We will choose $t=(d+1)\log n$, which is necessary for applying the following (fifth) reduction and the final algorithm.

Notice that this reduces exact learning degree-d s-sparse determinant-polynomials with black-box queries in $\mathrm{GF}(2)^{dn}$ to exact learning degree-d s-sparse determinant-polynomials with black-box queries in $\mathrm{GF}(2^t)^{dn}$. There are many non-adaptive learning algorithms of sparse polynomials over large fields. However, unfortunately, as we said before, degree-d s-sparse determinant-polynomials are degree-d $(d!s)$-sparse polynomials and $d!s=s^{\Omega(\log\log s)}$, which makes the time and query complexity of the algorithm super-polynomial. So, we need another reduction to reduce the number of monomials.

The fifth reduction reduces non-adaptive exact learning degree-d s-sparse determinant-polynomials with queries in $\mathrm{GF}(2^t)^{dn}$ to non-adaptive exact learning degree-d s-sparse (multilinear) polynomial over $\mathrm{GF}(2^t)$ with queries in $\mathrm{GF}(2^t)^n$. We simply choose, uniformly at random, dn elements $\alpha_{i,j}$ in $\mathrm{GF}(2^t)$ and substitute $y_{i,j}=\alpha_{i,j}x_i$. This changes each $\det(\mathcal{Y}^{(I)})$ to the monomial $(\det\Gamma^{(I)})\prod_{i\in I}x_i$ where for $I=\{i_1,i_2,\ldots,i_d\}$, $\Gamma^I_{k,j}=\alpha_{i_k,j}$. We then argue that whp, $\det\Gamma^{(I)}\neq 0$ and, therefore, we can find all the terms of the determinant-polynomials of the target.

Combining all the above, we reduce non-adaptive learning s-sparse polynomial to non-adaptive exact learning degree-d s-sparse polynomial with black-box that answer queries in $\mathrm{GF}(2^t)^n$. Now we use the algorithm of Ben-Or and Tiwari [3] with some modification to learn degree-d s-sparse multilinear polynomial over $\mathrm{GF}(2^t)$ with $2s$ queries. We show that to use Ben-Or and Tiwari algorithm, it is enough to have $t=(d+1)\log n$. This implies Theorem 1.

The following depicts the above reductions. Here, $\mathbb{P}_s$ is the class of s-sparse polynomials, $\mathbb{P}_{d,s}$ is the class of degree-d s-sparse polynomials, $\mathbb{DP}_{d,s}$ is the class of degree-d s-sparse determinant-polynomials, $\mathbb{P}_{d,s}[2^t]$ is the class of degree-d s-sparse multilinear polynomials over $\mathrm{GF}(2^t)$ and "qu. in" is an abbreviation of "queries in".

$$\begin{tikzcd}
 & & & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2^t)^{dn} \arrow[r] & \mathbb{P}_{d,s}[2^t] \text{ qu. in } \mathrm{GF}(2^t)^{n} \\
\mathbb{P}_s \arrow[r] & \mathbb{P}_{d,s} \arrow[r] & \mathbb{P}_{d,s}\ d\text{-monomials} \arrow[r] & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2)^{dn} \arrow[u] &
\end{tikzcd}$$

3 Definitions and Preliminary Results

We will denote by $\mathbb{P}_s$ the class of s-sparse polynomials over the Boolean variables $(x_1,\ldots,x_n)$ and $\mathbb{P}_{d,s} \subset \mathbb{P}_s$, the class of degree-$d$ s-sparse polynomials. For a power of two q, the classes $\mathbb{P}_s[q]$ is the class of s-sparse multilinear polynomials over the field $\mathrm{GF}(q)$ and $\mathbb{P}_{d,s}[q] \subset \mathbb{P}_s[q]$, the class of degree-$d$ s-sparse multilinear polynomials over $\mathrm{GF}(q)$.

Formally, let $\mathcal{S}_{n,\le d} = \cup_{i\le d}\mathcal{S}_{n,i}$, where $\mathcal{S}_{n,i} = \binom{[n]}{i}$ is the set of all i-subsets of $[n] = \{1,2,\ldots,n\}$. The class $\mathbb{P}_{d,s}$ (resp. $\mathbb{P}_{d,s}[q]$) is the class of all the polynomials of the form

$$\sum_{I\in S} a_I \prod_{i\in I} x_i$$

where $S \subseteq \mathcal{S}_{n,\le d}$, $|S| \le s$ and $a_I = 1$ for all I (resp. $a_I \in \mathrm{GF}(q)\backslash\{0\}$ for all I). The class $\mathbb{P}_s$ (resp. $\mathrm{P}_s[q]$) is $\mathbb{P}_{n,s}$ (resp. $\mathbb{P}_{n,s}[q]$).

Let $\{y_{j,i}\}_{i\in[n],j\in[d]}$ be nd variables. A degree-d s-sparse *determinant-polynomial* is a polynomial of the form

$$\sum_{I\in S} \det(\mathcal{Y}^{(I)}) \tag{1}$$

where $S \subseteq \mathcal{S}_{n,d}$, $|S| \le s$ and for $I = \{i_1,\ldots,i_d\} \subseteq [n]$, $\mathcal{Y}^{(I)}$ is the $d \times d$ matrix where $\mathcal{Y}^{(I)}_{j,k} = y_{j,i_k}$. We denote by $\mathbb{DP}_{d,s}$ the class of degree-d s-sparse determinant-polynomials.

Consider the operator $\phi_d : \mathbb{P}_{d,s} \to \mathbb{DP}_{d,s}$ defined as[2]

$$\phi_d f = \sum_{J\subseteq[d]} f\left(\sum_{j\in J} y_j\right) \tag{2}$$

where $y_j = (y_{j,1},\ldots,y_{j,n})$, $j \in [d]$. By Lemma 61 in [10], we have

$$\phi_d \sum_{I\in S}\prod_{i\in I} x_i = \sum_{I\in S,|I|=d} \det(\mathcal{Y}^{(I)}).$$

That is,

1. ϕ_d removes all the monomials of degree less than d from f.
2. It changes each monomial $\prod_{i\in I} x_i$ with $|I| = d$ to $\det(\mathcal{Y}^{(I)})$.
3. Given $\phi_d f$ represented as in (1), one can find f in linear time.
4. Every black-box query to $\phi_d f$ can be simulated by 2^d black-box queries to f.

[2] In [10] the sum contains $(-1)^{d-|J|}$. This disappears here since in GF(2), $-1 = 1$.

To see 3, notice that any row of $\mathcal{Y}^{(I)}$ uniquely gives I and therefore uniquely gives the monomial $\prod_{i\in I} x_i$.

We denote by $\mathbb{HP}_d$ the set of homogeneous polynomials F (over $\mathrm{GF}(2)$) of degree d over the variables $Y = \{y_{i,j}\}_{i\in[n],j\in[d]}$ where each monomial of F is of the form $y_{1,i_1}y_{2,i_2}\cdots y_{d,i_d}$ where $\{i_1, i_2, \ldots, i_d\} \in \binom{[n]}{d}$. Obviously, $\mathbb{DP}_{d,s} \subset \mathbb{HP}_d$

The following Lemma shows why we need the operator ϕ_d. Its proof is in Section 7.

Lemma 1. *For any integer power of two $t > 1$ there is an algorithm that runs in time $n\cdot poly(t, 2^d)$ and finds $N = \tilde{O}(2^{1.66d})t$ polynomial-time computable maps $M_i : \mathrm{GF}(2^t)^{dn} \to \mathrm{GF}(2)^{dn}$, $i \in [N]$, and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(2^t)$ such that for every $\beta \in \mathrm{GF}(2^t)^{dn}$ and every $F \in \mathbb{HP}_d$*

$$F(\beta) = \sum_{i=1}^{N} \alpha_i F(M_i(\beta)).$$

In particular, if $F \in \mathbb{DP}_d$ then a black-box query in $\mathrm{GF}(2^t)^{dn}$ to F can be simulated in time $n\cdot poly(N)$ by N black-box queries in $\mathrm{GF}(2)^{dn}$ to F.

In this paper, the representation that is used for elements in the Galois field $\mathrm{GF}(2^t)$ is $\mathrm{GF}(2)[w]/(g(w))$ for some irreducible polynomial $g \in \mathrm{GF}(2)[w]$ of degree t.

4 The Learning Algorithm

In this section, we give the learning algorithm for $\mathbb{P}_s$. Recall the reductions from Section 2.

$$\begin{tikzcd}
& & & \mathbb{DP}_{d,s}\text{ qu. in }\mathrm{GF}(2^t)^{dn} \arrow[r] & \mathbb{P}_{d,s}[2^t]\text{ qu. in }\mathrm{GF}(2^t)^{n} \\
\mathbb{P}_s \arrow[r] & \mathbb{P}_{d,s} \arrow[r] & \mathbb{P}_{d,s}\ d\text{-monomials} \arrow[r] & \mathbb{DP}_{d,s}\text{ qu. in }\mathrm{GF}(2)^{dn} \arrow[u] &
\end{tikzcd}$$

4.1 The Reduction Algorithms

In this subsection, we give the reductions. We will prove a lemma for each reduction.

Let $f(x_1, \ldots, x_n)$ be any Boolean function. A *p-zero projection* of f is a random function, $f(z) = f(z_1, \ldots, z_n)$ where each z_i is equal to x_i with probability p and is equal to 0 with probability $1-p$. The first reduction is Lemma 6 from [13]. The Lemma in [13] is stated for adaptive algorithm. The same proof holds for non-adaptive algorithms.

Lemma 2 ([13]). ($\mathbb{P}_s \to \mathbb{P}_{d,s}$). *Let $0 < p < 1$, $w = (s/\epsilon)^{\log(1/p)}\ln(16s)$ and*

$$D = \log\frac{s}{\epsilon} + \frac{\log s + \log\log s + 6}{\log(1/p)}.$$

Suppose there is a non-adaptive proper learning algorithm that exactly learns $\mathbb{P}_{d,s}$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with $O(w\cdot Q(D, 1/(16w))\cdot\log(1/\delta))$ queries, in time $w\cdot T(D, 1/(16w))\log(1/\delta)$, probability of success at least $1-\delta$ and accuracy $1-\epsilon$.

We now give the second reduction.

Lemma 3. *($\mathbb{P}_{d,s} \to \mathbb{P}_{d,s}$ d-monomials) Suppose there is a non-adaptive algorithm that for $f \in \mathbb{P}_{d,s}$ exactly learns the monomials of degree d of f with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}$ with $\sum_{j=0}^{d} Q(j,\delta/d)$ queries, time $O(d^2 s \max_j Q(j,\delta/d)) + \sum_{j=0}^{d} T(j,\delta/d)$ and probability of success at least $1-\delta$.*

Proof. Let $f \in \mathbb{P}_{d,s}$. Let f_i be the sum of all the monomials of f of degree i, and s_i the number of monomials of f_i. Let $A(d,\delta)$ be a non-adaptive algorithm that exactly learns the monomials of degree d of $f \in \mathbb{P}_{d,s}$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. We define an algorithm B as follows. First, algorithm B makes all the queries that all $A(i,\delta/d)$, $i \le d$, make. Let C_i be the set of queries that $A(i,\delta/d)$ makes, $i \le d$. Then B continues to run $A(d,\delta/d)$ with the answers of the queries in C_d and learns the monomials of degree d of f. Let f_d be the sum of those monomials. Then B continues to run $A(d-1,\delta/d)$ with $\{(a, f(a)+f_d(a)) | a \in C_{d-1}\}$. That is, for each query a of $A(d-1,\delta/d)$ we query f to find $f(a)$ and then return the answer $f(a)+f_d(a)$ to $A(d-1,\delta/d)$. Since $f+f_d$ is the sum of all the monomials of degree at most $d-1$ of f, $A(d-1,\delta/d)$ learns the monomials of degree $d-1$ of $f+f_d$, which are the monomials of degree $d-1$ of f. At the $d-i+1$-th stage, algorithm B continues to run $A(i,\delta/d)$ on

$$\left\{ \left(a, f(a) + \sum_{j=i+1}^{d} f_j(a) \right) \middle| \, a \in C_{d-1} \right\}$$

where f_j, $j \in \{i+1, i+2, \cdots, d\}$ is the sum of all the monomials of f of degree j. Since $g := f + \sum_{j=i+1}^{d} f_j$ is of degree i, $A(i,\delta/d)$ learns the monomials of degree i of g, which are the monomials of degree i of f. Notice that all the queries are all made for f, so the algorithm in non-adaptive.

It is clear that B exactly learns f, makes $\sum_{j=0}^{d} Q(j,\delta/d)$ queries, runs in time

$$O\left(\sum_{j=0}^{d} \left(\sum_{i=0}^{j} s_i \right) d \cdot Q(j,\delta/d) \right) + T(j,\delta/d) = O(d^2 s \max_j Q(j,\delta/d)) + \sum_{j=0}^{d} T(j,\delta/d),$$

and has probability of success at least $1-\delta$. □

The following is the third reduction.

Lemma 4. *($\mathbb{P}_{d,s}$ d-monomials $\to \mathbb{DP}_{d,s}$). Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ (with the matrix representation as in (1)) with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive exact learning algorithm that for $f \in \mathbb{P}_{d,s}$ learns the monomials of degree d of f with $2^d Q(d,\delta)$ queries in time $T(d,\delta) + O(2^d Q(d,\delta) n)$ and probability of success at least $1-\delta$.*

Proof. Let $A(d,\delta)$ be a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. We define an algorithm $B(d,\delta)$ as follows. Define $F = \phi_d f \in \mathbb{DP}_{d,s}$ and run $A(d,\delta)$ to learn F. Recall that ϕ_d removes all the monomials of degree less than d from f and changes each monomial $\prod_{i \in I} x_i$ with $|I| = d$ on f to $\det(\mathcal{Y}^{(I)})$. By item 4 in Section 3, every black-box query to F can be simulated by 2^d black-box queries to f. Given F in its matrix representation as in (1), we can find the degree-d monomials of f. See item 3 in Section 3. □

The following is the fourth reduction.

Lemma 5. *($\mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2)^{dt} \to \mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2^t)^{dt}$). Let t be a power of two. Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $Q(d,\delta)$ black-box queries in $\mathrm{GF}(2^t)^{dn}$, time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $2^{1.66d}tQ(d,\delta)$ black-box queries in $\mathrm{GF}(2)^{dn}$, time $T(d,\delta)+poly(t,2^d)Q(d,\delta)n$ and probability of success at least $1-\delta$.*

Proof. The result follows from Lemma 1. □

The following is the fifth reduction.

Lemma 6. *($\mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2^t)^{dt} \longrightarrow \mathbb{P}_{d,s}[2^t]$ queries in $\mathrm{GF}(2^t)^n$). Let $\ell = \lceil \log(2s/\delta)/(t-1)\rceil$. Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $Q(d,\delta)$ queries in $\mathrm{GF}(2^t)^n$ in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $\ell \cdot Q(d,\delta/(2\ell))$ queries in $\mathrm{GF}(2^t)^{dn}$ in time $O(\ell dnt)+\ell T(d,\delta/(2\ell))$ and probability of success at least $1-\delta$.*

Proof. Let $A(d,\delta)$ be a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Let $F(y_1,y_2,\ldots,y_d)\in\mathbb{DP}_{d,s}$ where $y_j=(y_{j,1},\ldots,y_{j,n})$, $j\in[d]$. We define an algorithm $B(d,\delta)$ as follows. Algorithm B uniformly at random chooses $dn\ell$ elements $\alpha_{j,i}^{(k)}\in\mathrm{GF}(2^t)$, $j\in[d]$, $i\in[n]$, and $k\in[\ell]$. Let $z_j^{(k)}=(\alpha_{j,1}^{(k)}x_1,\ldots,\alpha_{j,n}^{(k)}x_n)$, $j\in[d]$, $k\in[\ell]$ and $g^{(k)}(x_1,\ldots,x_n)=F(z_1^{(k)},z_2^{(k)},\ldots,z_d^{(k)})$. If $F=\sum_{I\in S}\det(\mathcal{Y}^I)$, then

$$g^{(k)}(x_1,\ldots,x_n)=\sum_{I\in S}\left(\left(\det\Gamma^{(k,I)}\right)\prod_{i\in I}x_i\right)$$

where for $I=\{i_1,\ldots,t_d\}$,

$$\Gamma^{(k,I)}=\begin{bmatrix}\alpha_{1,i_1}^{(k)} & \cdots & \alpha_{1,i_d}^{(k)}\\ \vdots & \ddots & \vdots\\ \alpha_{d,i_1}^{(k)} & \cdots & \alpha_{d,i_d}^{(k)}\end{bmatrix}.$$

Therefore, $\prod_{i\in I}x_i$ is a monomial of $g^{(k)}$ if and only if

1. $\det(\mathcal{Y}^I)$ is a term of F, and
2. $\det\Gamma^{(k,I)}\neq 0$.

Now notice that each $g^{(k)}$ is an s-sparse polynomial. Therefore, if we learn the monomials of $g^{(k)}$, $k\in[\ell]$, then we learn the terms $\det(\mathcal{Y}^I)$ of F for which $\det\Gamma^{(k,I)}\neq 0$. So, algorithm B runs $A(d,\delta/(2\ell))$ to learn all $g^{(k)}$, and from the monomials of all $g^{(k)}$ finds the terms of F.

The probability that B fails to learn all the monomials of F is equal to the probability that for some term $\det(\mathcal{Y}^I)$ of F, there is no $k\in[\ell]$ such that $\det\Gamma^{(I,k)}\neq 0$. Since $\alpha_{j,i}^{(k)}\in\mathrm{GF}(2^t)$ are uniformly random, this probability is at most

$$\begin{aligned} s\left(1-\left(1-\frac{1}{2^{dt}}\right)\left(1-\frac{1}{2^{(d-1)t}}\right)\cdots\left(1-\frac{1}{2^t}\right)\right)^{\ell} &\le s\left(\frac{1}{2^{dt}}+\frac{1}{2^{(d-1)t}}+\cdots+\frac{1}{2^t}\right)^{\ell}\\ &\le \frac{s}{2^{(t-1)\ell}}\le\frac{\delta}{2}. \end{aligned}$$

The probability that $A(d,\delta/(2\ell))$ fails to learn all $g^{(k)}$ is at most $\ell(\delta/(2\ell))=\delta/2$. This completes the proof. □

Now we show

Lemma 7. *($\mathbb{P}_{d,s}[2^t]$ queries in $\mathrm{GF}(2^t)^n$). Let $t=d'm$ be an integer where $d<d'=O(d)$ and $\log n<m=O(\log n)$. There is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $2s$ queries in $\mathrm{GF}(2^t)^n$ in time $poly(d,s)\cdot\tilde{O}(n)$.*

Proof. The result follows from the BCH decoding and the algorithm of Ben-Or and Tiwari [3], with a small modification. See the details in the Appendix. □

4.2 Query and Time Complexity

In this section, we prove,

Theorem 2. *Let $\beta>0$ be any real number. There is a non-adaptive proper learning algorithm for s-sparse polynomial with accuracy parameters $\epsilon=1/s^\beta$ and confidence parameter δ that makes*

$$q=O\left(\left(\frac{s}{\epsilon}\right)^{2.66+\frac{1}{\beta+1}+\frac{3.262}{\sqrt{\beta+1}}}\log n\log(1/\delta)\right)=O\left(\left(\frac{s}{\epsilon}\right)^{6.922}\log n\log(1/\delta)\right)$$

queries and runs in time $\tilde{O}(n)\cdot poly(s,1/\epsilon)\log(1/\delta)$.

Proof. Denote by $\lceil x\rceil_2$ the smallest power-of-two integer that is greater than or equal to x. Notice that $x\le\lceil x\rceil_2<2x$.

We choose $t=\lceil D+1\rceil_2\lceil\log n\rceil_2=O(D\log n)$, $\tau:=\log(1/p)$,

$$D=\log\frac{s}{\epsilon}+\frac{\log s+\log\log s+6}{\tau},$$

and

$$w=\left(\frac{s}{\epsilon}\right)^\tau\ln(16s),$$

where p will be determined later. We only need to know here that for this choice of p, we will have $D=\Theta(\log(s/\epsilon))$. By Lemma 7, there is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{D,s}[2^t]$ with $Q_1(D,\delta)=2s$ queries over $\mathrm{GF}(2^t)$ in time $T_1(D,\delta)=poly(D,s)\tilde{O}(n)$. By Lemma 6, there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{D,s}$ with $Q_2(D,\delta)=\tilde{O}(s)\log(1/\delta)$ queries over $\mathrm{GF}(2^t)$ in time $T_2(D,\delta)=poly(D,s)\tilde{O}(n)\log(1/\delta)$ and probability of success at least $1-\delta$. By Lemma 5, there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with

$$Q_3(D,\delta)=2^{1.66D}tQ_2(D,\delta)=\tilde{O}\left(\frac{s^{2.66+\frac{1.66}{\tau}+o_s(1)}}{\epsilon^{1.66}}\log n\log(1/\delta)\right)$$

queries in time

$$T_3(D,\delta)=T_2(D,\delta)+poly(t,2^D)Q_2(d,\delta)n=\tilde{O}(n)poly(s,1/\epsilon)\log(1/\delta)$$

and probability of success at least $1-\delta$. By Lemma 4 and 3, there is a non-adaptive exact learning algorithm that learns $\mathbb{P}_{D,s}$ with

$$Q_4(D,\delta) = D2^D Q_3(D,\delta/D) = \tilde{O}\left(\frac{s^{3.66+\frac{2.66}{\tau}+o_s(1)}}{\epsilon^{2.66}}\log n\log(1/\delta)\right)$$

queries in time $T_4(D,\delta)=\tilde{O}(n)\cdot poly(s,1/\epsilon)\log(1/\delta)$ and probability of success at least $1-\delta$. Now, by Lemma 2, there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with

$$Q(D,\delta) = O(w\cdot Q_4(D,1/(16w))\log(1/\delta)) = \tilde{O}\left(\frac{s^{3.66+\tau+\frac{2.66}{\tau}+o_s(1)}}{\epsilon^{2.66+\tau}}\log n\log(1/\delta)\right)$$

queries in time

$$T(D,\delta) = w\cdot T_4(D,1/(16w))\log(1/\delta) = \tilde{O}(n)\cdot poly(s,1/\epsilon)\log(1/\delta),$$

probability of success at least $1-\delta$ and accuracy $1-\epsilon$.

For $\epsilon = 1/s^\beta$ we choose[3] $\tau=\sqrt{2.66/(1+\beta)}$ and get

$$Q(D,\delta) = O\left(\left(\frac{s}{\epsilon}\right)^{2.66+\frac{1}{\beta+1}+\frac{3.262}{\sqrt{\beta+1}}}\log n\log(1/\delta)\right).$$

□

5 The Learning Algorithm for Small ϵ

In this section, we give a more query-efficient learning algorithm for s-sparse polynomials when $\epsilon<1/s^{0.752\log s}$. We prove

Theorem 3. *Let $\beta>0$ be any real number. There is a non-adaptive proper learning algorithm for s-sparse polynomials with accuracy parameters $\epsilon=1/s^\beta$ and confidence parameter δ that makes*

$$q = \left(s\log\frac{1}{\epsilon}\right)^{2\log s} s^{O(\log\log s)}\log n\log(1/\delta) = \left(\frac{s}{\epsilon}\right)^{\frac{2\log s+2\log\beta+O(\log\log s)}{\beta+1}}\log n\log(1/\delta)$$

queries and runs in time $\tilde{O}(qsn)$.

5.1 Technique

We will use the following result from [20]. See also [15].

Lemma 8. *Let $U_{n,s}$ be the set of all the assignments in $\{0,1\}^n$ of Hamming weight at least $n-\lfloor\log s\rfloor-1$. There is a non-adaptive exact learning algorithm for $\mathbb{P}_s$ that makes the black-box queries in $U_{n,s}$ and runs in time $sn(en/\log s)^{\log s+1}$.*

In particular,

1. *Let $f,g\in\mathbb{P}_s$ be two distinct s-sparse polynomials. There is $a\in U_{n,s}$ such that $f(a)\neq g(a)$.*

[3] This choice of τ minimizes the query complexity of the algorthm.

2. If $f \in \mathbb{P}_s$ depends on the variable x_i then there are two assignments $a, b \in U_{n,s}$ that differ only in the ith coordinate and $f(a) \neq f(b)$.

The above lemma follows from the fact that if f is of size $s > 1$, then there is $i \in [n]$ such that $f = f_1 x_i + f_2$ and $f_2 \not\equiv 0$. Then either[4] $f_2 = f_{x_i \leftarrow 0}$ is of size $\lfloor s/2 \rfloor$, or $f_1 = f_{x_i \leftarrow 0} + f_{x_i \leftarrow 1}$ is of size $\lfloor s/2 \rfloor$. See [20].

Item 2 follows from applying item 1 to $f(x)$ and $g(x) = f(1, x_2, \ldots, x_n)$.

In the first stage of our algorithm, we set each variable to 0 with probability $1 - 1/O(\log(s/\epsilon))$. This assignment removes monomials of size $D > \Omega((\log s)(\log(s/\epsilon))$. By Lemma 2, to be able to collect all the monomials of size at most $\log(s/\epsilon)$ of f, it is enough to take $O(\log s)$ such assignments. This reduced the non-adaptive learning of s-sparse polynomials to non-adaptive learning degree-d s-sparse polynomials, where $d = O((\log s)(\log s/\epsilon))$.

Now, let g be a degree-d s-sparse polynomial where $d = O((\log s)(\log s/\epsilon))$. The function $g(x)$ depends on at most $v = sd = O(s \log s \log(s/\epsilon))$ variables. We, uniformly at random, assign the n variables of g into $w = O(v^2)$ new variables $Y = \{y_1, y_2, \ldots, y_w\}$. Let $h(y)$ be the resulted function. With high probability, different relevant variables in g are assigned to different variables in Y. Now, assuming this event occurs, we run two algorithms. The first one learns $h(y)$ by the algorithm in Lemma 8. This takes $|U_{w,s}| \leq (ew/\log s)^{\log s} = (s \log(1/\epsilon))^{O(\log s)}$ queries. The second algorithm uses item 2 in Lemma 8 to find the relevant variable in $g(x)$ corresponding to each y_i (if any). To achieve that, for every two assignments in $U_{w,s}$ that differ in one coordinate, say i, we non-adaptively search for the relevant variable among all the variables that are assigned to y_i. Each search takes $O(\log n)$. This algorithm takes $w|U_{w,s}| \log n = (s \log(1/\epsilon))^{O(\log s)} \log n$ queries. This proves the Theorem.

5.2 The Algorithm

In this section, we give the algorithm and prove its correctness.

By Lemma 2 with $\log(1/p) = 1/\log(s/\epsilon)$, we have

Lemma 9. ($\mathbb{P}_s \to \mathbb{P}_{d,s}$). *Let $w = 2\ln(16s)$. Suppose there is a non-adaptive proper learning algorithm that exactly learns $\mathbb{P}_{d,s}$ with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. Then there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with $O(w \cdot Q(D, 1/(16w)) \log(1/\delta))$ queries where*

$$D = \left(\log \frac{s}{\epsilon}\right)(\log s + \log\log s + 7),$$

in time $w \cdot T(D, 1/(16w)) \log(1/\delta)$, probability of success at least $1 - \delta$ and accuracy $1 - \epsilon$.

The following is trivial,

Lemma 10. *There is a non-adaptive exact proper learning algorithm for $C = \{0, 1, x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ that makes $\log n + O(1)$ queries and runs in time $O(n \log n)$.*

We now prove

Lemma 11. *There is a proper exact learning algorithm for $\mathbb{P}_{d,s}$ with probability of success at least $1 - \delta$ that makes $q = O(d(2e(ds)^2/\log s)^{\log s} \log n \log(1/\delta))$ queries and runs in time $O(qn)$.*

[4] $f_{x_i \leftarrow 0}$ is f when we substitute 0 in x_i.

Proof. The algorithm draws uniformly at random a map $\phi : [n] \to [m]$ where $m = (2ds)^2$, and defines $F(x_1, \ldots, x_m) = f(x_{\phi(1)}, \ldots, x_{\phi(n)})$ and then exactly learns F using the algorithm in Lemma 8. Then, for every $i \in [m]$ and every $a, b \in U_{m,s}$ that differ in the ith coordinate, it learns, using the algorithm in Lemma 10, $f(\pi^{(i,a)})$ where

$$\pi_j^{(i,a)} = \begin{cases} a_{\phi(j)} & \phi(j) \neq i \\ x_j & \phi(j) = i \end{cases} .$$

Then the algorithm returns $F(y_1, \ldots, y_m)$ where $y_i = x_j$ if there is $a \in U_{m,s}$ such that $f(\pi^{(i,a)}) \in \{x_j, \bar{x}_j\}$ and $y_i = 0$ otherwise.

We now prove the correctness of the algorithm. Let $x_{r_1}, \ldots, x_{r_\ell}$, $\ell \le ds$, be the relevant variables of f. The probability that $\phi(r_1), \ldots, \phi(r_\ell)$ are distinct is at least $(1 - 1/m)(1 - 2/m) \cdots (1 - (\ell - 1)/m) \ge 7/8$. We now assume that this event occurs. In particular, $x_{\phi(r_1)}, \ldots, x_{\phi(r_\ell)}$ are the relevant variables of F.

Let $\phi(r_i) = t_i$. Let $a, b \in U_{m,s}$ be two assignments that differ in the t_i-th coordinate and $F(a) \neq F(b)$. Then $f(\pi^{(t_i,a)})$ is a non-constant function, and since $\phi(r_i) = t_i$, we have $f(\pi^{(t_i,a)}) \in \{x_{r_i}, \bar{x}_{r_i}\}$. Therefore, $y_{\phi(r_i)} = y_{t_i} = x_{r_i}$ and $F(y_1, \ldots, y_m) = f(y_{\phi(1)}, \ldots, y_{\phi(n)}) = f$.

The query complexity of the algorithm is $|U_{m,s}| = (em/\log s)^{\log s}$ for learning F and $d|U_{m,s}| = d(em/\log s)^{\log s} \log n$ for learning all $f(\pi^{(i,a)})$. This algorithm has a success probability of at least 7/8. Repeating the algorithm $O(\log(1/\delta))$ times gives the result. □

We are now ready to prove Theorem 3.

Proof. We first run the algorithm in Lemma 9. Then for each projection runs the algorithm in Lemma 11 with $d = D$. This gives the query complexity in the Theorem. □

6 Upper and Lower Bounds

In this section, we prove lower and upper bounds for learning sparse polynomials with unlimited computational power.

As we have seen in the previous sections, for constant confidence δ, the query complexity of the learning algorithms for $\epsilon = 1/s^\beta$ is of the form $(s/\epsilon)^{\gamma'} \log n$. In this section, we will study learning algorithms with query complexity

$$\left(\frac{s}{\epsilon}\right)^{\gamma(\beta)} \log n.$$

We denote by $\Gamma(\beta) = \min_A \gamma(\beta)$ the minimal possible value of $\gamma(\beta)$ among all the learning algorithms A of s-sparse polynomials with unlimited computational power. Formally,

$$\Gamma(\beta) = \min_A \lim_{n\to\infty} \frac{\log(q(A)/\log n)}{\log(s/\epsilon)}.$$

In particular, we may assume that s and $1/\epsilon$ are arbitrary small compared to n.

By Theorem 2 and 3, we already know the following bounds

$$\Gamma(\beta) \le \begin{cases} 2.66 + \frac{3.262}{\sqrt{\beta+1}} + \frac{1}{\beta+1} & \beta < 0.752 \log s \\ \\ \frac{2\log s + 2\log\beta + O(\log\log s)}{\beta+1} & \beta \ge 0.752 \log s \end{cases}$$

In this section, we prove the upper bound $\Gamma(\beta) \leq 1 + \min(1, \beta)/(\beta + 1)$. The above bounds gives the upper bound

$$\Gamma(\beta) \leq \begin{cases} 1 + \frac{\beta}{\beta+1} & \beta < 1 \\ 1 + \frac{1}{\beta+1} & 1 \leq \beta < 4.923 \\ \frac{\log \beta}{\beta+1} + \Theta\left(\frac{1}{\beta}\right) & \beta \geq 4.923 \end{cases} . \tag{3}$$

The exact bound when $\beta \geq 4.923$ is

$$\Gamma(\beta) \leq \min_{0.801<\eta<0.847} \frac{\eta+1}{\beta+1} + \left(1 + \eta^{-1}\right) H_2 \left(\frac{1}{(\beta+1)(1+\eta^{-1})}\right),$$

where $H_2(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function. In particular, $\Gamma(\beta) \leq 1.5$ for all β and $\Gamma(\beta) \to 0$ as $\beta \to 0$.

We then prove the lower bound,

$$\Gamma(\beta) \geq \begin{cases} \frac{1}{\beta+1} & 0 < \beta < 0.441 \\ 0.694 & 0.441 \leq \beta < 2.548 \\ \frac{\log \beta}{\beta+1} + \Theta\left(\frac{1}{\beta}\right) & \beta \geq 2.548 \end{cases} . \tag{4}$$

The exact bound when $\beta \geq 2.548$ is

$$\Gamma(\beta) \geq \frac{\beta \cdot H_2(1/\beta)}{\beta+1}.$$

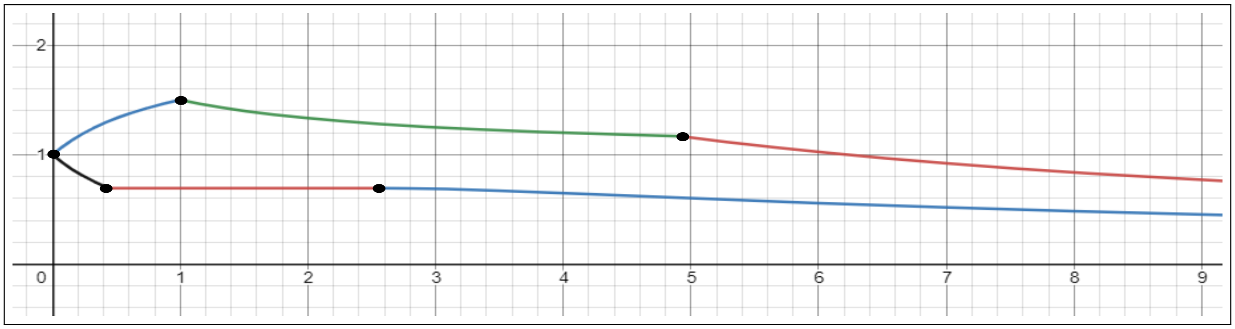

Figure 1: The lower and upper bounds of $\Gamma(\beta)$ for $0 \leq \beta \leq 9$.

Notice that our first algorithm in Section 4, for $\epsilon = 1/s^{O(1)}$, makes $(s/\epsilon)^\gamma \log n$ queries where $2.66 \leq \gamma \leq 6.922$ and the lower bound for this case is $(s/\epsilon)^{1/(\beta+1)} \log n = (s/\epsilon)^{\Theta(1)} \log n$. Therefore, our first algorithm is polynomial in the optimal query complexity and optimal in n.

The second algorithm in Section 5, for $\epsilon = 1/s^{\omega(\log s)}$, makes $(s/\epsilon)^{O((\log s + \log \beta)/\beta)} \log n = (s/\epsilon)^{o_\beta(1)} \log n$ queries and the lower bound is $(s/\epsilon)^{\beta H_2(1/\beta)/(\beta+1)} \log n = (s/\epsilon)^{\Theta(\log \beta/\beta)} \log n$. Therefore, the query complexity of the second algorithm is polynomial in the optimal query complexity when $\beta = s^{\Omega(1)}$.

6.1 Preliminary Results

In this section, we give some preliminary results.

Let $B_n = \{0,1\}^n$. The following result is well known. See, for example, [5].

Lemma 12. *For any $f \in \mathbb{P}_{d,s}$, we have $\mathbf{Pr}_{x\in B_n}[f(x)=1] \ge 2^{-d}$.*

Let $W_{n,s}$ be the set of all the assignments in $\{0,1\}^n$ of Hamming weight at least $n - \lfloor \log s \rfloor$. The following is from [20, 15]. See also Lemma 2 in [13].

Lemma 13. *For any $0 \ne f \in \mathbb{P}_s$ over n variables, there is an assignment $a \in W_{n,s}$ such that $f(a)=1$.*

The *p-product distribution* $\mathcal{D}_{n,p}$ is a distribution over $\{0,1\}^n$ where $\mathcal{D}_{n,p}(a) = p^{\mathrm{wt}(a)}(1-p)^{n-\mathrm{wt}(a)}$ and $\mathrm{wt}(a)$ is the Hamming weight of a.

The following two Lemmas are Lemma 3 and 4 in [13].

Lemma 14. *Let $p \ge 1/2$. For every $f \in \mathbb{P}_{n,d,s}$, $f \ne 0$, we have*

$$\mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)=1] \ge \left\{ \begin{array}{ll} p^{d-\lfloor \log s\rfloor}(1-p)^{\lfloor \log s\rfloor} & d \ge \lfloor \log s \rfloor \\ (1-p)^d & d < \lfloor \log s\rfloor \end{array} \right. .$$

In particular, if $d \ge 2\lfloor \log s\rfloor$, then for $p' = (d-\lfloor \log s\rfloor)/d$

$$\max_{p\ge 1/2} \mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)=1] = \mathbf{Pr}_{x\in\mathcal{D}_{n,p'}}[f(x)=1] \ge 2^{-H_2\left(\frac{\lfloor \log s\rfloor}{d}\right)d}$$

and if $d < 2\lfloor \log s\rfloor$, then for $p' = 1/2$

$$\max_{p\ge 1/2} \mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)=1] = \mathbf{Pr}_{x\in\mathcal{D}_{n,p'}}[f(x)=1] \ge 2^{-d}.$$

In particular, since $f(x) \ne g(x)$ is equivalent to $f(x)+g(x)=1$, we have

Lemma 15. *Let $p \ge 1/2$. For every $f, g \in \mathbb{P}_{n,d,s}$, $f \ne g$, we have*

$$\mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x) \ne g(x)] \ge \left\{ \begin{array}{ll} p^{d-\lfloor \log s\rfloor -1}(1-p)^{\lfloor \log s\rfloor+1} & d \ge \lfloor \log s \rfloor+1 \\ (1-p)^d & d < \lfloor \log s\rfloor+1 \end{array} \right. .$$

In particular, if $d \ge 2\lfloor \log s\rfloor+2$, then for $p' = (d-\lfloor \log s\rfloor-1)/d$

$$\max_{p\ge 1/2} \mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)\ne g(x)] = \mathbf{Pr}_{x\in\mathcal{D}_{n,p'}}[f(x)\ne g(x)] \ge 2^{-H_2\left(\frac{\lfloor \log s\rfloor+1}{d}\right)d}$$

and if $d < 2\lfloor \log s\rfloor+2$, then for $p' = 1/2$

$$\max_{p\ge 1/2} \mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)\ne g(x)] = \mathbf{Pr}_{x\in\mathcal{D}_{n,p'}}[f(x)\ne g(x)] \ge 2^{-d}.$$

In particular, for $p' = \max((d-\lfloor \log s\rfloor-1)/d, 1/2)$,

$$\max_{p\ge 1/2} \mathbf{Pr}_{x\in\mathcal{D}_{n,p}}[f(x)\ne g(x)] = \mathbf{Pr}_{x\in\mathcal{D}_{n,p'}}[f(x)\ne g(x)] \ge 2^{-H_2\left(\min\left(\frac{1}{2},\frac{\lfloor \log s\rfloor+1}{d}\right)\right)d}$$

We now prove.

Lemma 16. *There is a non-adaptive exact learning algorithm for $\mathbb{P}_{d,s}$ with probability of success of at least $1-\delta$ that runs in exponential time and makes*

$$q = 2^{H_2\left(\min\left(\frac{1}{2},\frac{\lfloor\log s\rfloor+1}{d}\right)\right)d}\ln\frac{|\mathbb{P}_{d,s}|}{\delta} = O\left(2^{H_2\left(\min\left(\frac{1}{2},\frac{\lfloor\log s\rfloor+1}{d}\right)\right)d}(ds\log n+\log(1/\delta))\right)$$

queries.

Proof. The algorithm draws q assignments S according to the distribution $\mathcal{D}_{n,p'}$, where $p' = \max((d-\lfloor\log s\rfloor-1)/d, 1/2)$, and then finds $h\in\mathbb{P}_{d,s}$ consistent with S on f.

By Lemma 15 and since $1-x\le e^{-x}$, the probability that $h\neq f$ is

$$\mathbf{Pr}[(\exists h\in\mathbb{P}_{d,s}, h\neq f)(\forall a\in S)h(a)=f(a)] \quad\le\quad |\mathbb{P}_{d,s}|\left(1-2^{-H_2\left(\min\left(\frac{1}{2},\frac{\lfloor\log s\rfloor+1}{d}\right)\right)d}\right)^q \le \delta.$$

□

The following are from [7, 8].

Lemma 17. *(Occam's lemma) Let C be a class of Boolean functions and $f\in C$. For $m=(1/\epsilon)\log(|C|/\delta)$ uniformly at random assignments $S=\{a^{(1)},\dots,a^{(m)}\}\subseteq\{0,1\}^n$, with probability at least $1-\delta$, any $h\in C$ that is consistent with f on S satisfies $\mathbf{Pr}_{x\in B_n}[f(x)\neq h(x)]\le\epsilon$.*

The following is from [19].

Lemma 18. *(Agnostic Learning) Let C be a class of Boolean functions, and f be any Boolean function. For $m=(1/2\epsilon^2)\ln(|C|/\delta)$ uniformly at random assignments $S=\{a^{(1)},\dots,a^{(m)}\}\subseteq\{0,1\}^n$, with probability at least $1-\delta$, all $h\in C$ satisfies*

$$|\mathbf{Pr}_{x\in B_n}[f(x)\neq h(x)]-\mathbf{Pr}_{x\in S}[f(x)\neq h(x)]|\le\epsilon.$$

The following is an information-theoretic lower bound.

Lemma 19. *Let C be a class of Boolean functions. Any exact learning algorithm for C must make at least $\log|C|$ queries.*

6.2 Upper Bounds

In this section, we prove the following three upper bounds.

Theorem 4. *There is a non-adaptive learning algorithm that runs in exponential time, makes*

$$\tilde{O}\left(\frac{s}{\epsilon^2}\right)\log n$$

queries, and learns $\mathbb{P}_s$ with a confidence probability of at least $2/3$.

In particular,

$$\Gamma(\beta)\le 1+\frac{\beta}{\beta+1}.$$

Proof. Let f be the target function. Consider $H=\mathbb{P}_{s,\log(3s/\epsilon)}$ and the following algorithm A. The algorithm A receives $m=O((1/\epsilon^2)\log|H|)=O((s/\epsilon^2)\log(s/\epsilon)\log n)$ uniformly at random assignments S and finds a function $h\in H$ that minimizes $\mathbf{Pr}_{x\in S}[h(x)\neq f(x)]$.

By Lemma 18 with probability at least 5/6, all $t\in H$ satisfies

$$|\mathbf{Pr}_{x\in B_n}[f(x)\neq t(x)]-\mathbf{Pr}_{x\in S}[f(x)\neq t(x)]|\le \epsilon/3. \tag{5}$$

Let g be the sum of all the monomials of f of size at most $\log(3s/\epsilon)$. Then $g(x)\in H$ and $\mathbf{Pr}_{x\in B_n}[f(x)\neq g(x)]\le s2^{-\log(3s/\epsilon)}\le\epsilon/3$. By (5), with probability at least 5/6, $\mathbf{Pr}_{x\in S}[f(x)\neq g(x)]\le 2\epsilon/3$. Therefore, $\mathbf{Pr}_{x\in S}[f(x)\neq h(x)]\le 2\epsilon/3$ and by (5), with probability at least 2/3, $\mathbf{Pr}_{x\in B_n}[f(x)\neq h(x)]\le\epsilon$. □

Theorem 5. *There is a non-adaptive learning algorithm that runs in exponential time, makes*

$$\tilde{O}\left(\frac{s^2}{\epsilon}\right)\log n$$

queries, and learns $\mathbb{P}_s$ *with a confidence probability of at least* 2/3.

In particular,

$$\Gamma(\beta)\le 1+\frac{1}{\beta+1}.$$

Proof. Let f be the target function. Let $H=\mathbb{P}_{d,s}$ where $d=8s\log(2s/\epsilon)\log(8s)$. Consider the following algorithm A. It first takes a $(1/(8s\log(2s/\epsilon)))$-zero projection $f(z)$, takes $m=O((1/\epsilon)\log|H|)=O(\log(s/\epsilon)(\log s)(s^2/\epsilon))$ uniformly at random assignments S, and finds a function $h\in H$ that is consistent with $f(z)$ on S.

The probability that all the monomials of size at most $\log(2s/\epsilon)$ of f are also of $f(z)$ is at least

$$\left(1-\frac{1}{8s\log(2s/\epsilon)}\right)^{s\log(2s/\epsilon)}\ge\frac{7}{8}.$$

The probability that $f(z)\in H$ is at least

$$1-s\left(1-\frac{1}{8s\log(2s/\epsilon)}\right)^{8s\log(2s/\epsilon)\log(8s)}\ge 1-se^{-\log(8s)}\ge\frac{7}{8}.$$

Therefore, with probability at least 3/4, $f(z)\in H$ and $\mathbf{Pr}_{x\in B_n}[f(x)\neq f(z)]\le\epsilon/2$. Now by Lemma 17, with probability at least 11/12, $\mathbf{Pr}_{x\in B_n}[h(x)\neq f(z)]\le\epsilon/2$. Therefore, with probability at least 2/3, $\mathbf{Pr}_{x\in B_n}[h(x)\neq f(x)]\le\epsilon$. □

Now Theorems 3, 4, and 5 give the upper bound in (3).

Theorem 6. *Let* $\epsilon=1/s^\beta$ *for* $\beta>1$. *Let*

$$\begin{aligned}\gamma(\beta) &= \min_{0.801<\eta<0.847}\frac{\eta+1}{\beta+1}+\left(1+\eta^{-1}\right)H_2\left(\frac{1}{(\beta+1)(1+\eta^{-1})}\right)\\ &= \frac{\log\beta}{\beta+1}+O\left(\frac{1}{\beta+1}\right).\end{aligned}$$

There is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with probability of success at least $1-\delta$ and accuracy $1-\epsilon$, runs in exponential time, and makes

$$q = \tilde{O}\left(\left(\frac{s}{\epsilon}\right)^{\gamma(\beta)+o(1)}\right)(\log n)\log(1/\delta)$$

queries.

In particular,

$$\Gamma(\beta) \le \min_{0.801<\eta<0.847} \frac{\eta+1}{\beta+1} + \left(1+\eta^{-1}\right) H_2\left(\frac{1}{(\beta+1)(1+\eta^{-1})}\right).$$

Proof. We use the reduction in Lemma 2 with Lemma 16. Let $\lambda = \log(1/p) \ge 0$. The number of queries is

$$q = \tilde{O}\left(\left(\frac{s}{\epsilon}\right)^{\lambda} 2^{H_2\left(\frac{\lfloor \log s\rfloor+1}{D}\right)D} Ds\right)(\log n)\log(1/\delta),$$

where

$$D = \log\frac{s}{\epsilon} + \frac{\log s + \log\log s + 6}{\lambda} \ge 2\log 2 + 2.$$

For $\epsilon = 1/s^{\beta}$, we get

$$q = \left(\frac{s}{\epsilon}\right)^{\lambda+\frac{1}{\beta+1}+\left(1+\frac{1}{\lambda(\beta+1)}\right)H_2\left(\frac{1}{\beta+1+\lambda^{-1}}\right)+o(1)}(\log n)\log(1/\delta).$$

For $\lambda = \eta/(\beta+1)$, where $0.801 \le \eta \le 0.847$, we get

$$q = \left(\frac{s}{\epsilon}\right)^{\frac{\eta+1}{\beta+1}+\left(1+\eta^{-1}\right)H_2\left(\frac{1}{(\beta+1)(1+\eta^{-1})}\right)+o(1)}(\log n)\log(1/\delta).$$

□

6.3 Lower Bounds

In this section, we give three lower bounds that prove the lower bound in (4). We remind the reader that the parameters s and $1/\epsilon$ are arbitrary small compared to n.

We first give the following information-theoretic lower bound.

Theorem 7. *Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\log\frac{1}{\epsilon}\right)s\log n\right)$$

queries. In particular, when $\epsilon = 1/s^{\beta}$, the bound is

$$\tilde{\Omega}\left(\left(\frac{s}{\epsilon}\right)^{\frac{1}{\beta+1}}\right)\log n \ \ \textit{and} \ \ \Gamma(\beta) \ge \frac{1}{\beta+1}.$$

Proof. Consider the class $C = \mathbb{P}_{\log(1/(2\epsilon)),s}$. Consider a (randomized) non-adaptive learning algorithm A_R for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ and accuracy ϵ. Then A_R is also a (randomized) non-adaptive learning algorithm for C. Since by Lemma 12, any two distinct functions in C have distance 2ϵ, A_R exactly learns[5] C with a confidence probability of at least $2/3$. By Yao's minimax principle, there is a deterministic non-adaptive exact learning algorithm A_D with the same query complexity as A_R that learns at least $(2/3)|C|$ functions in $|C|$. By Lemma 19, the query complexity of A_D is at least $\log|C| - 2$. Since

$$\log|C| = \log \binom{\binom{n}{\log(1/(2\epsilon))}}{s} = \Omega\left(\left(\log\frac{1}{\epsilon}\right) s \log n\right),$$

the result follows. □

We now show the following.

Theorem 8. *Let $\epsilon = 1/s^\beta$, $\beta > 2$. Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\frac{s}{\epsilon}\right)^{\frac{\beta\cdot H_2(1/\beta)}{\beta+1}} \log n\right)$$

queries.

In particular, for $\beta > 2$,

$$\Gamma(\beta) \ge \frac{\beta \cdot H_2(1/\beta)}{\beta+1} \ge \frac{\log\beta}{\beta+1}.$$

Proof. Let $t = \log(1/\epsilon) - \log s - 2 \ge \log s - 2$ and $r = \log s$. Let W be the set of all pairs (I, J) where I and J are disjoint sets, $I \cup J = [t+r]$, $|I| \ge t$, and $|J| = t + r - |I| \le r$. For every $(I, J) \in W$, define $f_{I,J} = \prod_{i\in I} x_i \prod_{j\in J}(1 + x_j)$. For $k \in [n]\backslash[t+r]$ define $f_{I,J,k} = x_k \cdot f_{I,J}$. Consider the set C of all such functions. First notice that $f_{I,J,k} \in C \subset \mathbb{P}_{t+r,s} \subseteq \mathbb{P}_s$ and, by Lemma 12, $\mathbf{Pr}[f_{I,J,k} = 1] \ge 2^{-(t+r+1)} = 2^{-\log(1/\epsilon)+1} = 2\epsilon$. Furthermore, since for $(I_1, J_1, k_1) \ne (I_2, J_2, k_2)$ the degree of $f_{I_1,J_1,k_1} + f_{I_2,J_2,k_2}$ is $\log(1/\epsilon) - 1$, we also have

$$\mathbf{Pr}[f_{I_1,J_1,k_1} \ne f_{I_2,J_2,k_2}] \ge 2\epsilon.$$

Therefore, any learning algorithm for $\mathbb{P}_s$ (with accuracy ϵ and confidence $2/3$) is a learning algorithm for C and thus is an exact learning algorithm for C.

Consider now a (randomized) non-adaptive exact learning algorithm A_R for C with a success probability of at least $2/3$ and accuracy ϵ. By Yao's minimax principle, there is a deterministic non-adaptive exact learning algorithm A_D that for uniformly at random $f \in C$, with a probability at least $2/3$, A_D returns f. We will show that A_D must make more than $q = (1/20)w \log N$ queries where $N = n - (t + r)$ and

$$w = |W| = \sum_{i=0}^{r} \binom{t+r}{i}.$$

Now since, $r \le (t + r)/2 + 1$,

$$w = \sum_{i=0}^{\log s} \binom{\log\frac{1}{\epsilon} - 2}{i} \ge \tilde{\Omega}\left(2^{H_2\left(\frac{\log s}{\log(1/\epsilon)}\right)\log(1/\epsilon)}\right) = \tilde{\Omega}\left(\left(\frac{1}{\epsilon}\right)^{H_2(1/\beta)}\right) = \tilde{\Omega}\left(\left(\frac{s}{\epsilon}\right)^{\frac{\beta\cdot H_2(1/\beta)}{\beta+1}}\right)$$

[5] Just take the function in C closest to the output of A_R

we get the result.

To this end, suppose for the contrary, A_D makes q queries. Let $S = \{a^{(1)}, \ldots, a^{(q)}\}$ be the set of queries that A_D makes. For every $(I, J) \in W$, let $S_{I,J} = \{a \in S | f_{I,J}(a) = 1\}$. Since for any two distinct $(I_1, J_1), (I_2, J_2) \in W$, we have $f_{I_1,J_1} \cdot f_{I_2,J_2} = 0$, the sets $\{S_{I,J}\}_{(I,J)\in W}$ are disjoint. Let $f = f_{I',J',k'}$ be uniformly at random function in C. We will show that, with probability at least $4/5$, A_D fails to learn f, which gives a contradiction.

Since

$$E_{(I,J)\in W}[|S_{I,J}|] = \frac{\sum_{(I,J)\in W} |S_{I,J}|}{|W|} = \frac{q}{w} = (1/20)\log N,$$

by Markov's bound with probability at least $9/10$, we have $|S_{I',J'}| \le (1/2)\log N$. Since for $(I, J) \ne (I', J')$, $f_{(I',J',k')}(S_{I,J}) = \{0\}$, the only queries that are relevant for learning k' in f are the ones in $S_{I',J'}$. Therefore, it is enough to show that, if $|S_{I',J'}| \le (1/2)\log N$, then with probability at least $9/10$, the queries in $S_{I',J'}$ fail to learn k'. Since the algorithm is deterministic and the number of possible answers to the queries in $S_{I',J'}$ is $2^{(1/2)\log N} = \sqrt{N}$, the number of possible distinct outputs of the algorithm is at most $\sqrt{N}$. Since k' is drawn uniformly at random and can take N possible values, the probability that the algorithm returns k' is less than or equal to $\sqrt{N}/N = 1/\sqrt{N} \le 1/10$. Therefore, the algorithm fails to output k' with probability at least $9/10$. This completes the proof. □

We now show

Theorem 9. *Let $\epsilon = 1/s^\beta$, $0.44 \le \beta \le 2.61$. Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\frac{s}{\epsilon}\right)^{0.694} \log n\right)$$

queries.

Proof. Let η be such that

$$\frac{1+\beta\eta}{1+\beta} = \frac{5+\sqrt{5}}{10} \approx 0.7236.$$

Then $0.0955 \le \eta \le 0.618$. Also

$$\frac{1+\beta\eta}{\beta(1-\eta)} = \frac{1}{(1+\beta)/(1+\beta\eta) - 1} = \frac{3+\sqrt{5}}{2} \approx 2.618. \tag{6}$$

Let

$$t = \log s + (2\eta - 1)\log(1/\epsilon) + 3 = (1 + \beta(2\eta - 1))\log s + 3 \tag{7}$$

and

$$r = (1-\eta)\log(1/\epsilon) - 3 = \beta(1-\eta)\log s - 3.$$

We first show that $t \ge 3$. By 6, $1 + \beta\eta > \beta(1-\eta)$ and therefore $(1 + \beta(2\eta - 1)) > 0$. Thus, by 7, $t \ge 3$.

Let

$$\sigma = 8\epsilon^{1-\eta}s = 8s^{1-\beta(1-\eta)}.$$

Since, by 6, $\beta(1-\eta) = (1+\beta\eta)/2.618 \le (1+2.61\cdot 0.618)/2.618 < 1$, we have $\sigma \ge 8$.

Let W be the set of all r-sets $J \subseteq I := [t+r]$. For every $J \in W$, define $f_J = \prod_{i\in I\backslash J} x_i \prod_{j\in J}(1+x_j)$. Let U be the set of all σ-sets $\mathcal{J} = \{(J_1,t_1),\ldots,(J_\sigma,t_\sigma)\}$ where $J_1,\ldots,J_\sigma \subseteq I$ are distinct r-sets and $t_k \in [n]\backslash[t+r]$. Let

$$f_{\mathcal{J}}(x) = \sum_{k=1}^{\sigma} x_{t_k} f_{J_k} = \sum_{k=1}^{\sigma}\left(x_{t_k} \prod_{i\in I\backslash J_k} x_i \prod_{i\in J_k}(1+x_i)\right).$$

We first show that $f_{\mathcal{J}}$ is well-defined. That is, it is possible to choose σ distinct r-sets $J_1,\ldots,J_\sigma \subset I$. This is possible because

$$\begin{aligned}\binom{t+r}{r} &= \binom{(1+\beta\eta)\log s}{\beta(1-\eta)\log s - 3} \\ &= s^{(1+\beta\eta)H\left(\frac{\beta(1-\eta)}{1+\beta\eta}\right)-o(1)} = s^{0.9594(1+\beta\eta)-o(1)} \qquad (8)\end{aligned}$$

and

$$\sigma = 8s^{1-\beta(1-\eta)} = 8s^{1-0.382(1+\beta\eta)} < s^{0.626(1+\beta\eta)} < s^{0.9594(1+\beta\eta)-o(1)} = \binom{t+r}{r}.$$

Consider the class $C = \{f_{\mathcal{J}} | \mathcal{J} \in U\}$. The number of monomials of $f_{\mathcal{J}}(x)$ is $\sigma 2^r = s$ and therefore $C \subseteq \mathbb{P}_s$. Since the terms of $f_{\mathcal{J}}$ are disjoint, we have $\mathbf{Pr}[f_{\mathcal{J}} = 1] = \sigma 2^{-(t+r)-1} = 4\epsilon^{1-\eta}s\cdot(\epsilon^\eta/s) = 4\epsilon$. Notice that the distance between every two functions in C can be as small as $2\cdot 2^{-(t+r)} = 2\epsilon^\eta/s = 2\epsilon^{\eta+\beta}$, which may take values less than ϵ. Therefore, the argument used in Theorem 8 will not hold here.

Consider now a (randomized) non-adaptive learning algorithm A_R for C with a success probability of at least 2/3 and accuracy ϵ. By Yao's minimax principle, there is a deterministic non-adaptive learning algorithm A_D that for a uniformly at random target function $f_{\mathcal{J}} \in C$, with probability at least 2/3, A_D returns a hypothesis $h^{(\mathcal{J})}$ that is ϵ-close to $f_{\mathcal{J}}$. Consider the deterministic algorithm A'_D that runs A_D and outputs $\mathcal{J}'$ where $f_{\mathcal{J}'}$ is the closest function in C to $h^{(\mathcal{J})}$. Since $h^{(\mathcal{J})}$ is ϵ-close to both $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$, $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$ are 2ϵ-close. We will show in the sequel that if $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$ are 2ϵ-close, then $|\mathcal{J}\cap\mathcal{J}'| \ge \sigma/2$. This shows that A'_D returns a σ-set $\mathcal{J}'$ that, with probability at least 2/3, $|\mathcal{J}\cap\mathcal{J}'| \ge \sigma/2$ where $f_{\mathcal{J}}$ is the target function. We now show that if A'_D makes less than $q = (1/100)w\log N$ queries where $w = \binom{t+r}{r}$ and $N = n-(s+t)$ then, with probability at least 2/3, algorithm A'_D fails to output such $\mathcal{J}'$. Now, since, by (8),

$$\binom{t+r}{r} = s^{0.9594(1+\beta\eta)-o(1)} = \left(\frac{s}{\epsilon}\right)^{0.9594\frac{1+\beta\eta}{\beta+1}-o(1)} = \left(\frac{s}{\epsilon}\right)^{0.6942-o(1)},$$

the result follows.

To this end, suppose A'_D makes less than q queries. Let $S = \{a^{(1)},\ldots,a^{(q)}\}$ be the queries that A'_D makes. For every $J \in W$, let $S_J = \{a \in S | f_J(a) = 1\}$. Since for any two distinct $J_1, J_2 \in W$, we have $f_{J_1}f_{J_2} = 0$, the sets $\{S_J\}_{J\in W}$ are disjoint. Since

$$E_{J\in W}[|S_J|] = \frac{\sum_{J\in W}|S_J|}{|W|} = \frac{q}{w} = \frac{\log N}{100},$$

by Markov's bound, at least 49/50 fraction of the r-subsets J of $[t+r]$ satisfy $|S_J| \le (1/2)\log n$. Now, for a uniformly at random target function $f_{\mathcal{J}} = f_{\{(J_1,t_1),\ldots,(J_\sigma,t_\sigma)\}} \in C$, let X_i be an indicator random variable that is equal to 1 if $|S_{J_i}| > (1/2)\log N$. Then $\mathbf{E}[X_i] \le 1/50$. Let $X = X_1 + \cdots + X_\sigma$. Since $\mathbf{E}[X] \le \sigma/50$, by Markov's bound, with probability at least 4/5, at least $9\sigma/10$ of the J_is in $\mathcal{J}$ satisfy $|S_{J_i}| < (1/2)\log N$. Assume, wlog, $|S_{J_i}| < (1/2)\log N$ for $i \le 9\sigma/10$. As in Theorem 8, the algorithm A'_D can find each t_i, $i \le 9\sigma/10$ with probability at most $1/\sqrt{N} < 1/20$. Since the t_is are drawn iid and uniformly at random, by Chernoff bound, with probability at least 4/5, the algorithm A'_D fails to find $\sigma/2$ of the t_is for $i \le 9\sigma/10$. Therefore, with probability at least 2/3, A'_D fails to return a σ-set $\mathcal{J}'$ such that $|\mathcal{J} \cap \mathcal{J}'| \ge \sigma/2$.

It remains to show that if $f_{\mathcal{J}}$ is 2ϵ-close to $f_{\mathcal{J}'}$, then $|\mathcal{J} \cap \mathcal{J}'| \ge \sigma/2$. Suppose, for the contrary, $|\mathcal{J} \cap \mathcal{J}'| < \sigma/2$. Suppose, wlog, $\mathcal{J} = \{(J_1, t_1), \ldots, (J_\sigma, t_\sigma)\}$ and

$$\mathcal{J}' = \{(J_1, t_1), \ldots, (J_\ell, t_\ell), (J_{\ell+1}, t'_{\ell+1}), \ldots, (J_\mu, t'_\mu), (J'_{\mu+1}, t'_{\mu+1}), \ldots, (J'_\sigma, t'_\sigma)\}$$

where J_i, J'_k, $i \in [\sigma]$, $k \ge \mu+1$ are distinct r-subsets of $[r+t]$ and $t'_i \ne t_i$ for $\mu \ge i > \ell$. Then

$$f_{\mathcal{J}} + f_{\mathcal{J}'} \;=\; \sum_{k=\ell+1}^{\mu} (x_{t_k} + x_{t'_k}) f_{J_k} + \sum_{k=\mu+1}^{\sigma} x_{t_k} f_{J_k} + \sum_{k=\mu+1}^{\sigma} x_{t'_k} f_{J'_k}.$$

Since $f_{J_i}, f_{J'_k}$, $i \in [\sigma]$, $k \ge \mu+1$ are pairwise disjoint and $2\sigma - \mu - \ell > \sigma/2$, we have

$$\begin{aligned}\mathbf{Pr}[f_{\mathcal{J}} \ne f_{\mathcal{J}'}] &= \mathbf{Pr}[f_{\mathcal{J}} + f_{\mathcal{J}'} = 1] = (2\sigma - \mu - \ell)2^{-(t+r)-1} \\ &> \frac{\sigma}{2} 4\epsilon^{1-\eta} s \cdot (\epsilon^\eta / s) = 2\epsilon.\end{aligned}$$

A contradiction. □

7 Proof of Lemma 1

Recall that $\mathbb{HP}_d$ is the set of homogeneous polynomial F of degree d over the variables $Y = \{y_{i,j}\}_{i\in[n],j\in[d]}$ where each monomial of F is of the form $y_{1,i_1} y_{2,i_2} \cdots y_{d,i_d}$ where $\{i_1, i_2, \ldots, i_d\} \in \binom{[n]}{d}$. Throughout this section, q will be a power of two. All the arithmetic operations in the field $\mathrm{GF}(q^t)$ have time complexity $poly(t, \log q)$. Therefore, each arithmetic operation will be considered as one unit-time.

Let $d > 1$ be an integer. We write $\mathrm{GF}(q^t) \xrightarrow{N}_d \mathrm{GF}(q)$ if there are Nd linear maps $M_{i,j} : \mathrm{GF}(q^t) \to \mathrm{GF}(q)$, $i \in [N]$, $j \in [d]$ and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(q^t)$ such that for every $a_1, a_1, \ldots, a_d \in \mathrm{GF}(q^t)$

$$\prod_{j=1}^{d} a_j = \sum_{k=1}^{N} \left(\alpha_k \prod_{j=1}^{d} M_{k,j}(a_j) \right). \tag{9}$$

We say that $\mathrm{GF}(q^t) \xrightarrow{N}_d \mathrm{GF}(q)$ *in time* T if such maps can be found in time T.

We first prove

Lemma 20. *If* $\mathrm{GF}(2^t) \xrightarrow{N}_d \mathrm{GF}(2)$ *in time* T*, then we can find maps* $M^*_k : \mathrm{GF}(2^t)^{dn} \to \mathrm{GF}(2)^{dn}$*,* $k \in [N]$ *in time* $O(Tn)$ *that are computable in time* $O(dnt)$ *such that, for every* $F \in \mathbb{HP}_d$ *and every* $\beta \in \mathrm{GF}(2^t)^{dn}$*, we have*

$$F(\beta) = \sum_{i=1}^{N} \alpha_i F(M^*_i(\beta)).$$

Proof. Since $\mathrm{GF}(2^t)\xrightarrow{N}_d\mathrm{GF}(2)$ in time T, we can find Nd linear maps $M_{i,j} : \mathrm{GF}(2^t) \to \mathrm{GF}(2)$, $i \in [N]$, $j \in [d]$ and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(2^t)$ such that for every $a_1, a_1, \ldots, a_d \in \mathrm{GF}(q^t)$, (9) holds.

We define $M_k^*((y_{j,i})_{i\in[n],j\in[d]}) = (M_{k,j}(y_{j,i}))_{i\in[n],j\in[d]}$. Let F be any function in $\mathbb{HP}_d$. Suppose

$$F((y_{j,i})_{i\in[n],j\in[d]}) = \sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I \prod_{j=1}^{d} y_{j,i_j}\right)$$

where $S \subseteq \binom{[n]}{d}$ and $\gamma_I \in \mathrm{GF}(2)$ for all $I \in S$. Then for every $\beta \in \mathrm{GF}(2^t)^{dn}$, by 9, we have

$$\begin{aligned}
F(\beta) &= \sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I \prod_{j=1}^{d} \beta_{j,i_j}\right)\\
&= \sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I \sum_{k=1}^{N}\left(\alpha_k \prod_{j=1}^{d} M_{k,j}(\beta_{j,i_j})\right)\right)\\
&= \sum_{k=1}^{N}\left(\alpha_k \sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I\left(\prod_{j=1}^{d} M_{k,j}(\beta_{j,i_j})\right)\right)\right)\\
&= \sum_{k=1}^{N}\alpha_k F((M_{k,j}(\beta_{j,i}))_{i\in[n],j\in[d]}) = \sum_{k=1}^{N}\alpha_k F(M_k^*(\beta)).
\end{aligned}$$

□

In particular, to prove Lemma 1, it is enough to prove.

Claim 1. *For any integer power of two $t > 1$, we have $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ in time poly$(t, 2^d)$.*

We now prove some Lemmas which leads to this result.

Using exhaustive search for the linear maps $M_{i,j}$ and $\{\alpha_i\}_{i\in N}$, we have

Lemma 21. *If $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ then $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ in time $q^{O(tdN)}$.*

The following four Lemmas are easy to prove

Lemma 22. *If $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ in time T then for every $N' \geq N$, $\mathrm{GF}(q^t)\xrightarrow{N'}_d\mathrm{GF}(q)$ in time T.*

Lemma 23. *If $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ in time T then for every $d' \leq d$, $\mathrm{GF}(q^t)\xrightarrow{N}_{d'}\mathrm{GF}(q)$ in time $O(T)$.*

Lemma 24. *If $\mathrm{GF}(q^{t_1t_2})\xrightarrow{N_2}_d\mathrm{GF}(q^{t_1})$ in time T_1 and $\mathrm{GF}(q^{t_1})\xrightarrow{N_1}_d\mathrm{GF}(q)$ in time T_2 then*

$$\mathrm{GF}(q^{t_1t_2})\xrightarrow{N_1N_2}_d\mathrm{GF}(q)$$

in time $O(T_1 + T_2 + N_1N_2t_1t_2)$.

Lemma 25. *If* $\mathrm{GF}(q^t)\xrightarrow{N_1}_{d_1}\mathrm{GF}(q)$ *in time* T_1 *and* $\mathrm{GF}(q^t)\xrightarrow{N_2}_{d_2}\mathrm{GF}(q)$ *in time* T_2 *then*

$$\mathrm{GF}(q^t)\xrightarrow{N_1N_2}_{d_1+d_2}\mathrm{GF}(q)$$

in time $O(T_1+T_2+N_1N_2)$.

In particular, by Lemmas 23 and 25, we get

Lemma 26. *If* $\mathrm{GF}(q^t)\xrightarrow{N}_{d'}\mathrm{GF}(q)$ *in time* T *then* $\mathrm{GF}(q^t)\xrightarrow{N^{\lceil d/d'\rceil}}_{d}\mathrm{GF}(q)$ *in time* $O(dT/d'+N^{\lceil d/d'\rceil})$.

Now we prove

Lemma 27. *If* $q\ge d(t-1)$, *then* $\mathrm{GF}(q^t)\xrightarrow{d(t-1)+1}_{d}\mathrm{GF}(q)$ *in time poly*(dt).

Proof. Let $f^{(1)},\dots,f^{(d)}\in\mathrm{GF}(q)[x]$ be arbitrary d polynomials of degree $t-1$ where $f^{(i)}=\sum_{j=0}^{t-1}f_j^{(i)}x^j$. Choose an element $\beta\in\mathrm{GF}(q^t)$ such that $\mathrm{GF}(q^t)\cong\mathrm{GF}(q)[\beta]$. Then $a_1:=f^{(1)}(\beta),\dots,$ $a_d:=f^{(d)}(\beta)$ are arbitrary d elements in $\mathrm{GF}(q^t)$. So it is enough to show that $\prod_{i=1}^d f^{(i)}(\beta)$ can be expressed as in (9) with $N=d(t-1)+1$.

Let $f=\prod_{i=1}^d f^{(i)}(x)=f_0+f_1x+\ldots+f_{d(t-1)}x^{d(t-1)}$. One way to express the coefficients of f as functions of $f_j^{(i)}$, $i\in[d]$, $j=0,1,\dots,d(t-1)$, is the following: First, we have $f_{d(t-1)}=\prod_{i=1}^d f_t^{(i)}$. Then substitute $d(t-1)$ distinct elements $\eta_1,\dots,\eta_{d(t-1)}$ of the field $\mathrm{GF}(q)$ in $\prod_{i=1}^d f^{(i)}(x)$ and interpolate to find coefficients $f_0,\dots,f_{d(t-1)-1}$. This shows that for every $i=0,1,\dots,d(t-1)$, there are $\omega_{i,0},\omega_{i,1},\dots,\omega_{i,d(t-1)}$ independent of $f_j^{(i)}$, $i\in[d]$, $j=0,1,\dots,d(t-1)$ such that

$$f_i=\omega_{i,0}\prod_{k=1}^d f_t^{(k)}+\sum_{j=1}^{d(t-1)}w_{i,j}f(\eta_j)=\omega_{i,0}\prod_{k=1}^d f_t^{(k)}+\sum_{j=1}^{d(t-1)}w_{i,j}\prod_{k=1}^d f^{(k)}(\eta_j).$$

Then

$$\prod_{i=1}^d f^{(i)}(x)=\sum_{i=0}^{d(t-1)}f_ix^i=\left(\sum_{i=0}^{d(t-1)}\omega_{i,0}x^i\right)\prod_{k=1}^d f_t^{(k)}+\sum_{j=1}^{d(t-1)}\left(\sum_{i=0}^{d(t-1)}w_{i,j}x^i\right)\prod_{k=1}^d f^{(k)}(\eta_j).$$

Let

$$\alpha_i=\sum_{i=0}^{d(t-1)}w_{i,j}\beta^i$$

for $j=0,1,\dots,d(t-1)$. Then

$$\prod_{i=1}^d f^{(i)}(\beta)=\alpha_0\prod_{k=1}^d f_t^{(k)}+\sum_{j=1}^{d(t-1)}\alpha_j\prod_{k=1}^d f^{(k)}(\eta_j).$$

Now recall that $f^{(i)}(\beta)$, $i\in[d]$ are d (arbitrary) elements in $\mathrm{GF}(q^t)$. Since $\eta_j\in\mathrm{GF}(q)$ we have $f_i^{(k)}$ and $f^{(k)}(\eta_j)$ are linear functions from $\mathrm{GF}(q^t)$ to $\mathrm{GF}(q)$. This implies the result. □

The following lemma proves Claim 1 for a power of two $t=O(\log d)$.

Lemma 28. *Let c be a constant. Let t be a power of two such that $t < c\log d$. We have* $\mathrm{GF}(2^t)\xrightarrow{2^{1.66d}}_d\mathrm{GF}(2)$ *in time poly(2^d).*

Proof. By Lemma 27, $\mathrm{GF}(2^2)\xrightarrow{3}_2 GF(2)$ in time $O(1)$. By Lemma 26, $\mathrm{GF}(2^2)\xrightarrow{N_1=3^{\lceil d/2\rceil}}_d GF(2)$ in time $O(N_1)$. Now for any i, by Lemma 27, $\mathrm{GF}(2^{2^{i+1}})\xrightarrow{2^{2^i}+1}_{2^{2^i}} GF(2^{2^i})$ in time $poly(2^{2^i})$. By Lemma 26, $\mathrm{GF}(2^{2^{i+1}})\xrightarrow{N_i}_d\mathrm{GF}(2^{2^i})$ in time $poly(2^{2^i})+O(N_i)$ where

$$N_i=(2^{2^i}+1)^{\left\lceil d/2^{2^i}\right\rceil}.$$

By Lemma 24, we have $\mathrm{GF}(2^t)\xrightarrow{N}_d\mathrm{GF}(2)$ in time $poly(2^t,2^{\log^2 d}N)=poly(2^d)$ where

$$\begin{array}{rcl} N & = & 3^{\lceil d/2\rceil}5^{\lceil d/4\rceil}\cdots(2^{t/2}+1)^{\lceil d/2^{t/2}\rceil} \\ & \le & 2^{O(\log^2 d)}2^{\left(\frac{\log 3}{2}+\frac{\log 5}{4}+\cdots\right)d} \\ & \le & 2^{O(\log^2 d)}\cdot 2^{1.65994d}\le 2^{1.66d}. \end{array}$$

□

The following lemma will be frequently used to prove Claim 1 for larger values of d.

Lemma 29. *Let t and t' be powers of 2, where $2\log(dt) > t' \ge \log(dt)$. Then* $\mathrm{GF}(2^t)\xrightarrow{dt/t'}_d\mathrm{GF}(2^{t'})$ *in time poly(dt).*

Proof. We have $q=2^{t'}\ge dt\ge d(t/t'-1)$. By Lemma 22 and 27, the result follows. □

The following lemma proves Claim 1 for a power of two $t=d^{O(1)}$.

Lemma 30. *Let c be a constant. Let t be a power of two such that $t<d^c$. We have* $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ *in time poly(2^d).*

Proof. By Lemma 29, $\mathrm{GF}(2^t)\xrightarrow{dt/t'}_d\mathrm{GF}(2^{t'})$ in time $poly(dt)$ for some power of two t', where $t'\le 2(c{+}1)\log d$. By Lemma 28, $\mathrm{GF}(2^{t'})\xrightarrow{2^{1.66d}}_d\mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 24, $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ in time $poly(2^d)$. □

The following lemma proves Claim 1 for a power of two $t=2^{d^{O(1)}}$.

Lemma 31. *Let c be a constant. Let t be a power of two such that $t<2^{d^c}$. We have* $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ *in poly$(t,2^d)$.*

Proof. By Lemma 29, $\mathrm{GF}(2^t)\xrightarrow{dt/t'}_d\mathrm{GF}(2^{t'})$ in time $poly(dt)$ for some power of two t', where $t'\le d^{c+2}$. By Lemma 30, $\mathrm{GF}(2^{t'})\xrightarrow{\tilde{O}(2^{1.66d})t'}_d\mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 24, $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ in time $poly(t,2^d)$. □

The following is from [11], Corollary 17 item 7

Lemma 32. *For any $q\ge d$ and t, we have* $\mathrm{GF}(q^t)\xrightarrow{O(d^4t)}_d\mathrm{GF}(q)$.

By Lemma 32 and Lemma 21 we get

Lemma 33. *For any $q \ge d$ and t, we have* $\mathrm{GF}(q^t)\xrightarrow{O(d^4t)}_d\mathrm{GF}(q)$ *in time* $q^{O(t^2d^5)}$.

The following lemma proves Claim 1 for a power of two $t > 2^{d^{12}}$.

Lemma 34. *Let $t \ge 2^{d^{12}}$ be a power of two. We have* $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ *in time $poly(t, 2^d)$.*

Proof. Let $2d > 2^\ell \ge d$ power of two. By Lemma 29, we have $\mathrm{GF}(2^t)\xrightarrow{dt/t'}_d\mathrm{GF}(2^{t'})$ in time $poly(dt)$ for a power of two $2\log(dt) > t' \ge \log(dt)$. Then again, by Lemma 29, we have $\mathrm{GF}(2^{t'})\xrightarrow{dt'/t''}_d\mathrm{GF}(2^{t''})$ in time $poly(dt')$ for a power of two $2\log(dt') > t'' \ge \log(dt')$. By Lemma 33, $\mathrm{GF}(2^{t''})\xrightarrow{O(d^4t''/\ell)}_d\mathrm{GF}(2^\ell)$ in time

$$(2^\ell)^{O((t''/\ell)^2d^5))} \le 2^{O(d^6\log^2\log t)} \le 2^{O(\log^{1/2} t\log^2\log t)} < t.$$

By Lemma 30, $\mathrm{GF}(2^\ell)\xrightarrow{\tilde{O}(2^{1.66d})\ell}_d\mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 24, $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ in time $poly(t, 2^d)$. □

Now Claim 1 follows immediately from Lemma 30 and 34.

References

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

[2] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000. `doi:10.1145/337244.337257`.

[3] Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 301–309. ACM, 1988. `doi:10.1145/62212.62241`.

[4] Francesco Bergadano, Nader H. Bshouty, and Stefano Varricchio. Learning multivariate polynomials from substitution and equivalence queries. *Electron. Colloquium Comput. Complex.*, (8), 1996. URL: `https://eccc.weizmann.ac.il/eccc-reports/1996/TR96-008/index.html`.

[5] Laurence Bisht, Nader H. Bshouty, and Hanna Mazzawi. On optimal learning algorithms for multiplicity automata. In Gábor Lugosi and Hans Ulrich Simon, editors, *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, volume 4005 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2006. `doi:10.1007/11776420\_16`.

[6] Avrim Blum and Mona Singh. Learning functions of k terms. In Mark A. Fulk and John Case, editors, *Proceedings of the Third Annual Workshop on Computational Learning Theory, COLT 1990, University of Rochester, Rochester, NY, USA, August 6-8, 1990*, pages 144–153. Morgan Kaufmann, 1990. URL: `http://dl.acm.org/citation.cfm?id=92620`.

[7] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Inf. Process. Lett.*, 24(6):377–380, 1987. doi:10.1016/0020-0190(87)90114-1.

[8] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989. doi:10.1145/76359.76371.

[9] Nader H. Bshouty. On learning multivariate polynomials under the uniform distribution. *Inf. Process. Lett.*, 61(6):303–309, 1997. doi:10.1016/S0020-0190(97)00021-5.

[10] Nader H. Bshouty. Testers and their applications. *Electron. Colloquium Comput. Complex.*, page 11, 2012. URL: https://eccc.weizmann.ac.il/report/2012/011.

[11] Nader H. Bshouty. Testers and their applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 327–352. ACM, 2014. doi:10.1145/2554797.2554828.

[12] Nader H. Bshouty. Almost optimal testers for concise representations. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:156, 2019. URL: https://eccc.weizmann.ac.il/report/2019/156.

[13] Nader H. Bshouty. Almost optimal proper learning and testing polynomials. *CoRR*, abs/2202.03207, 2022. URL: https://arxiv.org/abs/2202.03207, arXiv:2202.03207.

[14] Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM J. Comput.*, 31(6):1909–1925, 2002. doi:10.1137/S009753979732058X.

[15] Michael Clausen, Andreas W. M. Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991. doi:10.1016/0304-3975(91)90157-W.

[16] Arne Dür and Johannes Grabmeier. Applying coding theory to sparse interpolation. *SIAM J. Comput.*, 22(4):695–704, 1993. doi:10.1137/0222046.

[17] Paul Fischer and Hans Ulrich Simon. On learning ring-sum-expansions. *SIAM J. Comput.*, 21(1):181–192, 1992. doi:10.1137/0221014.

[18] Lisa Hellerstein and Rocco A. Servedio. On PAC learning algorithms for rich boolean function classes. *Theor. Comput. Sci.*, 384(1):66–76, 2007. doi:10.1016/j.tcs.2007.05.018.

[19] Michael J. Kearns, Robert E. Schapire, and Linda Sellie. Toward efficient agnostic learning. *Mach. Learn.*, 17(2-3):115–141, 1994. doi:10.1007/BF00993468.

[20] Ron M. Roth and Gyora M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM J. Comput.*, 20(2):291–314, 1991. doi:10.1137/0220019.

[21] Robert E. Schapire and Linda Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. Comput. Syst. Sci.*, 52(2):201–213, 1996. doi:10.1006/jcss.1996.0017.

[22] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.

Appendix: Proof of Lemma 7

In this Section, we prove
Lemma 7. *Let $t = d'm$ be an integer where $d < d' = O(d)$ and $\log n < m = O(\log n)$. There is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $2s$ queries in $\mathrm{GF}(2^t)^n$ in time poly$(d,s) \cdot \tilde{O}(n)$.*

Proof. The proof is the same as in [3].

Let $w \in \mathrm{GF}(2^t)$ such that $\mathrm{GF}(2^t) \simeq \mathrm{GF}(2^m)[w]$. The minimal polynomial $p \in \mathrm{GF}(2^m)[x]$ of w is of degree $d'-1 \geq d$. Each element in $\mathrm{GF}(2^t)$ can be represented as $\gamma_0 + \gamma_1 w + \cdots + \gamma_{d'-1}w^{d'-1}$ where $\gamma_i \in \mathrm{GF}(2^m)$ for all i.

Let

$$f(x) = \sum_{i=1}^{s} a_i M_i,$$

where $M_i = x_{r_{i,1}} x_{r_{i,2}} \cdots x_{r_{i,d_i}}$ and $d_i \leq d$ for all $i \in [s]$. Here, we assume that the number of monomials in f is exactly s. We will show later the case when the size is less than s.

We choose n distinct elements $\alpha_1, \alpha_2, \cdots, \alpha_n$ in $\mathrm{GF}(2^m)$. This is possible because $2^m \geq n$. The queries of the algorithm are

$$u_i = ((w-\alpha_1)^i, (w-\alpha_2)^i, \ldots, (w-\alpha_n)^i), \quad i = 0, 1, 2, \ldots, 2s-1.$$

Let $v_i = f(u_i)$.

We now show how to find M_i, $i \in [s]$, and then find the coefficients a_i, $i \in [s]$.

We first give some observations. Let $m_i = M_i(u_1)$ and $\Lambda(x) = \prod_{i\in[s]}(x - m_i) = \sum_{i=0}^{s} \lambda_i x^i$. Notice that $M_i(u_j) = m_i^j$ and $\Lambda(m_i) = 0$ for every i. Now, for every $\ell = 0, 1, 2, \ldots, s-1$

$$\begin{aligned}
0 &= \sum_{i=1}^{s} a_i m_i^{\ell} \Lambda(m_i) \\
&= \sum_{i=1}^{s} a_i m_i^{\ell} \sum_{j=0}^{s} \lambda_j m_i^j \\
&= \sum_{j=0}^{s} \lambda_j \sum_{i=1}^{s} a_i m_i^{\ell+j} \\
&= \sum_{j=0}^{s} \lambda_j \sum_{i=1}^{s} a_i M_i(u_{\ell+j}) \\
&= \sum_{j=0}^{s} \lambda_j v_{\ell+j}.
\end{aligned}$$

Since $\lambda_s = 1$, we get the linear equation $V\lambda = v$, where

$$V = \begin{pmatrix} v_0 & v_1 & v_2 & \cdots & v_{s-1} \\ v_1 & v_2 & v_3 & \cdots & v_s \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{s-1} & v_s & v_{s+1} & \cdots & v_{2s-1} \end{pmatrix}, \lambda = \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{s-1} \end{pmatrix} \text{ and } v = \begin{pmatrix} -v_s \\ -v_{s+1} \\ \vdots \\ -v_{2s-1} \end{pmatrix}.$$

Ben-Or and Tiwari show in [3] that if f has s monomials, then V is a non-singular matrix. Then we can find $\lambda = V^{-1}v$ and therefore $\Lambda(x)$. By factoring $\Lambda(x)$, we get $m_i = M_i(u_1)$. Let $m_i = m_{i,0} + m_{i,1}w + \cdots + m_{i,d'-1}w^{d'-1}$ where $m_{i,j} \in \mathrm{GF}(2^m)$ for all j. Now since $m_i = M_i(u_1) = (w - \alpha_{r_{i,1}}) \cdots (w - \alpha_{r_{i,d_i}})$ and the minimal polynomial of w is of degree $d' - 1 \geq d \geq d_i$, we have

$$m_{i,0} + m_{i,1}x + \cdots + m_{i,d'-1}x^{d'-1} = (x - \alpha_{r_{i,1}}) \cdots (x - \alpha_{r_{i,d_i}}).$$

So by factoring $m_{i,0} + m_{i,1}x + \cdots + m_{i,d'-1}x^{d'-1}$, we get $\alpha_{r_{i,1}}, \ldots, \alpha_{r_{i,d_i}}$ and therefore $M_i(x)$. To find the coefficients a_i, we solve the linear equation

$$f(u_i) = \sum_{i=1}^{s} a_i M_i(u_i) = v_i, \quad i = 0, 1, \ldots, s-1.$$

This finishes the case when the number of monomials is s. When the number of monomial is at most s, then it is known that the number of monomials is equal to the maximum r such that the upper left $r \times r$ sub-matrix of V is non-singular. So we find r and continue as above. □