RANDOMIZED INTERPOLATION AND APPROXIMATION OF SPARSE POLYNOMIALS

YISHAY MANSOUR*

Abstract. We present a randomized algorithm that interpolates a sparse polynomial in polynomial time in the bit complexity model. The algorithm can be also applied to approximate polynomials that can be approximated by sparse polynomials (the approximation is in the L_2 norm).

1. Introduction. We consider the problem of interpolating a sparse polynomial with integer coefficients. The interpolation model is the following. The algorithm can query the polynomial P on any input x, and receives the value $P(x)$ with some finite accuracy. The accuracy parameter determines the number of significant bits that are given from the value of $P(x)$, which enables to truncate $P(x)$ after that number of significant bits. The running time of the algorithm is the number of bit operations performed.

The parameters of the sparse polynomial are: n – the number of variables, t – an upper bound on the number of non-zero coefficients, d – an upper bound on the degree of a single variable and L – an upper bound on the absolute value of the largest coefficient. A t-sparse polynomial is a polynomial with at most t non-zero coefficients. The number of bits required to express a t-sparse polynomial is $O(t(\log L + n\log d)) = poly(n, t, \log d, \log L)$[1].

Our main result is a randomized algorithm that interpolates a t sparse polynomial in time $poly(n, t, \log d, \log L)$. In the analysis of the algorithm we count the number of operations performed, but since the accuracy that our algorithm requires is very small (logarithmic in the above parameters), the running time remains $poly(n, t, \log d, \log L)$ in the bit complexity model.

An interesting application of our algorithm is finding an *approximation* by a sparse polynomial. The approximation is in the L_2 norm, i.e. the expected error squared is small. The main result is that if a polynomial P can be approximated by a t-sparse polynomial Q, such that the power of $P - Q$ is bounded by ϵ, then our algorithm can find a t-sparse polynomial Q' such that the power of $P - Q'$ is bounded by $O(\epsilon)$. (The *power* of a polynomial is the sum of the squares of its coefficients.) Our algorithm achieves this by finding only the "large" coefficients of P, rather than finding all of the coefficients of P. This notion of approximation is especially interesting when the values of the variables have approximately unit magnitude. We believe that this notion of approximation is especially interesting in our framework, since if we allow the value of the variable be, for example, 2, then the output of the polynomial may have already d bits.

---

* Dept. of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. Part of the research done while the author was at Aiken Computation Laboratory, Harvard University, and IBM - T. J. Watson Research Center.

[1] We denote by $poly(a)$ a function which is bounded by a polynomial in a, i.e. $a^{O(1)}$

Interpolation of a univariate polynomial can be performed in time $O(d \log d)$, using the Fast Fourier Transform (FFT). The FFT does not use the information about the sparsity of the polynomial, therefore, its running time is polynomial in d, rather than t and $\log d$. Since the FFT algorithm does not consider the sparsity of the polynomial, for multivariate polynomials where the number of possible coefficients is exponential, its running time is exponential.

Previous works on sparse multi-variate polynomial interpolation were mainly in the algebraic model, where each operation requires one unit of time. When considering those works in the bit complexity model one has to take into account the number of bits which were given to the polynomial, and also the size of the result of the polynomial and the required accuracy on the result. Zippel [Zip79] gave a probabilistic algorithm that runs in time $poly(n, t, d)$. His algorithm evaluates the polynomial at points of size $\Theta(\log ntd)$ bits and requires $\Theta(d \log ntd + \log L)$ bits, which implies that the running time in the bit complexity model is $poly(n, t, d, \log L)$.

Grigoriev and Karpinski [GK87] showed a deterministic NC algorithm to compute a matching, in the case that the permanent is sparse. (The permanent can be viewed as a generic multivariate polynomial.) Ben-Or and Tiwari [BOT88] gave a deterministic algorithm to compute a sparse multivariate polynomial that runs in time $poly(n, t, \log d)$. The algorithm evaluates the polynomial at points of size $O(t \log n)$ bits and requires accuracy of $O(td \log n + \log L)$ bits. Therefore, the running time of their algorithm in the bit complexity model requires time $poly(n, t, d, \log L)$. Grigoriev, Karpinski and Singer [GKS90b] showed how to interpolate a rational function with $O(nt^t)$ points, which is independent of the degree d ! However in the bit complexity model the running time is $poly(n, t^t, d, \log L)$. Algorithms for interpolation of sparse polynomials over finite field appears in [GKS90a, Kar89].

The work here extends the technique developed in [GL89, KM91] for the Fourier transform over the hypercube. The extension handles a ring of integers modulo an integer (i.e. Z_d). After we show how to handle Z_d, we use the previous techniques and extend the result to handle Z_d^n.

Remark: In this work we are interested in the bit complexity model. In the analysis we use the algebraic model and count the number of operations. By guaranteeing that the precision that our algorithm requires is small, we make sure that its running time in both models is similar.

The paper is organized as follows. Section 2 mentions a few known facts about the Chernoff bounds, discrete Fourier transform and defines the black box model. Section 3 has the interpolation to univariate sparse polynomials with small coefficients, which is extended in Section 4 to handle arbitrary coefficients. The precision accuracy that the algorithm requires is discussed in Section 5. Section 6 deals with the approximation of a real valued function by sparse polynomials. Section 7 extends the case of univariate polynomials to multivariate polynomials.

2. Preliminaries. Chernoff bounds: In the analysis of our randomized algorithms we use Chernoff bounds. The following theorem formalizes the bounds in a way which would be most useful for us. (See [HR89].)

THEOREM 2.1. *(Chernoff bound) Let $X_1, \ldots, X_m$ be i.i.d with $E[X_i] = p$, $|X_i| \leq H$ and let $S_m = X_1 + \cdots + X_m$. If $m \geq \frac{2H^2}{\epsilon^2} \ln \frac{2}{\delta}$ then,*

$$Pr\left[\left| \frac{S_m}{m} - p \right| > \epsilon \right] \leq \delta$$

Complex numbers: The magnitude of a complex number $z = a+ib$ is $\|z\| = \sqrt{a^2+b^2}$. Let $\omega_d = e^{\frac{2\pi}{d}i}$ be a d-th root of unity. Recall that $\|\omega_d^k\| = 1$, for any d and k.
Discrete Fourier Transform: We list here a few known facts about the Discrete Fourier Transform. Consider a polynomial $P(x) = \sum_{j=0}^{d-1} a_j x^j$. Each coefficient of the polynomial satisfies,

$$a_j = \frac{1}{d} \sum_{k=0}^{d-1} P(\omega_d^k)\omega_d^{-kj}.$$

Parseval identity relates the sum of the squares of the coefficients to the values of the polynomial as follows,

$$\sum_{j=0}^{d-1} a_j^2 = \frac{1}{d}\sum_{k=0}^{d-1} P(\omega_d^k)P(\omega_d^{-k}) = \frac{1}{d}\sum_{k=0}^{d-1} \|P(\omega_d^k)\|^2.$$

For any integer k not divisible by d,

$$\sum_{m=0}^{d-1} (\omega_d^k)^m = 0.$$

The Black Box Model: Our algorithm assumes that it has a black box that evaluates the polynomial P. The algorithm evaluates the polynomial at roots of unity of the form ω_d^k. The interaction between the algorithm and the black box is the following.

To evaluate the polynomial P at ω_d^k, the algorithm gives the black box the pair (d,k). Denote by $\rho(k,d)$ the output of the black box on input (d,k).

The black box has *precision accuracy* ϵ if for any input (d,k), the absolute error is at most ϵ, i.e.,

$$\|P(\omega_d^k) - \rho(d,k)\| \leq \epsilon.$$

Let $H = \max_k\{\|P(\omega_d^k)\|\}$. The number of bits required for precision accuracy ϵ is $O(\log H + \log \frac{1}{\epsilon})$.

3. Univariate polynomials. In this section we discuss interpolation of t-sparse polynomials with integer coefficients. The result is based on a searching technique that we develop to search in the space of the possible coefficients. The search technique is similar to the one in [GL89, KM91].

Given a polynomial $P(x)$ we split it into two polynomials. One that includes the coefficients of odd degrees (denoted by $P_{1,1}$), while the other includes the coefficients of even degrees (denoted by $P_{0,1}$). Note that $P_{0,1}(x) = (P(x) + P(-x))/2$ and $P_{1,1}(x) =$

$(P(x) - P(-x))/2$. In a similar way we define $P_{\alpha,\ell}(x)$, where $0 \le \alpha \le 2^\ell - 1$ and $0 \le \ell \le \log d$, as follows,

$$P_{\alpha,\ell}(x) = \sum_{j:\, e_j \bmod 2^\ell = \alpha} a_{e_j} x^{e_j}$$

The algorithm works as follows. It keeps a list of the current candidates polynomials, $P_{\alpha,\ell}$. The algorithm takes a polynomial $P_{\alpha,\ell}$ off the list and tests if it is identically zero. If the polynomial is identically zero no action is taken. (This implies that we terminate the search for coefficients of exponents e, $e = \alpha \bmod 2^\ell$, since all those coefficients are zero.) Otherwise we continue recursively searching, i.e. we add to the list of candidates polynomials the polynomials $P_{\alpha+2^\ell,\ell+1}$ and $P_{\alpha,\ell+1}$. When $\ell = \log d$ then we have reached a non-zero coefficient. (Namely the polynomial $P_{\alpha,\log d}(x) = a_\alpha x^\alpha$.)

Since there are at most t non-zero coefficients and for each coefficient we perform $\log d$ recursive calls, the total number of recursive calls is bounded by $O(t \log d)$.

We are not able to compute the values of the polynomials that we create *exactly*, but rather we are able to approximate them very well. One way to overcome this is to approximate the value of a polynomial at randomly selected points, and if at some point it is more than 1/2 then we assume that it is a non-zero polynomial. Here we choose an alternative way to test for zero polynomials that is based on approximating the power of the polynomial. Since the polynomial has integer coefficients, if it is a non-zero polynomial then it has power at least one. We approximate the power of the polynomial and if the approximation is more than 1/2 we assume that the polynomial is non-zero. We prefer the method that is based on approximating the power since it extends more naturally to the case of large coefficients and to the case of approximation by sparse polynomials.

3.1. The main procedure. In this subsection we outline the main idea of how to recover the coefficients of a t-sparse polynomial $P(x) = \sum_{j=1}^{t} a_{e_j} x^{e_j}$, whose coefficients are integers. The basic strategy is to recover the exponents e_j bit after bit, starting from the least significant bit.

Let d be the smallest power of two greater than $\max_j\{e_j\}$, i.e. $d = 2^{\lceil \log(1+\max_j\{e_j\}) \rceil}$. Consider the sets $S_\ell = \{e_j \bmod 2^\ell \ : \ 1 \le j \le t\}$, for $0 \le \ell \le \log d$. The set S_ℓ includes the ℓ least significant bits of the exponents e_j. Clearly the size of a set S_ℓ is bounded by t, and the set S_0 includes only zero. We show how to compute the set $S_{\ell+1}$ given the set S_ℓ.

From the definition of S_ℓ it follows that if $\alpha \notin S_\ell$ then $P_{\alpha,\ell}(x) \equiv 0$ while if $\alpha \in S_\ell$ then $P_{\alpha,\ell}(x) \not\equiv 0$. One step in the construction is testing if a polynomial is identically zero. Assume that we are given such a procedure $ZERO(P_{\alpha,\ell})$ that returns $TRUE$ if and only if $P_{\alpha,\ell} \equiv 0$ (later we will show how to implement it).

Given the set S_ℓ we construct the set $S_{\ell+1}$ as follows. For each $\alpha \in S_\ell$ we create two integers: $\alpha_0 = \alpha$ and $\alpha_1 = \alpha + 2^\ell$. For each α_σ, $\sigma \in \{0,1\}$, we test $ZERO(P_{\alpha_\sigma,\ell})$. If $ZERO(P_{\alpha_\sigma,\ell})$ is $FALSE$ we add α_σ to $S_{\ell+1}$. Since the set $S_{\log d}$ includes all the exponents of P, when we recover $S_{\log d}$ we have found all the exponents of P.

CLAIM 3.1. *The algorithm constructs each S_ℓ correctly.*

Proof. The proof is by induction on ℓ. Initially S_0 includes only zero, so it is correct. Assume that we constructed S_ℓ correctly. We have to show two properties of $S_{\ell+1}$: that any element added should have been added and that every element of $S_{\ell+1}$ was added at some point.

An element $\beta = \alpha_\sigma$, $\sigma \in \{0,1\}$, is added to $S_{\ell+1}$ only if $ZERO(P_{\beta,\ell+1})$ is $FALSE$. This implies that $\sum_{j:e_j \bmod 2^{\ell+1}=\beta} a_{e_j}^2 \neq 0$. Therefore, there exists some coefficient a_{e_j}, such that $e_j \bmod 2^{\ell+1} = \beta$, which is not zero.

For the second part assume that $\beta \in S_{\ell+1}$. Let $\beta \bmod 2^\ell = \alpha$. By the induction hypothesis $\alpha \in S_\ell$. When we handle α, in one of the two cases we test $\alpha_\sigma = \beta$. Since $\beta \in S_{\ell+1}$, $ZERO(P_{\beta,\ell+1}) = ZERO(P_{\alpha_\sigma,\ell+1})$ is $FALSE$. Hence β was added to $S_{\ell+1}$. □

LEMMA 3.2. *The number of calls to the procedure $ZERO$ is bounded by $O(t \log d)$*

□

Proof. There are $\log d$ sets S_ℓ. The size of each set S_ℓ is bounded by t. For each element $\alpha \in S_\ell$ we perform two calls, $ZERO(\alpha_0, \ell)$ and $ZERO(\alpha_1, \ell)$. □

So far we have an algorithm that requires an oracle for computing $P_{\alpha,\ell}(x)$, and to the procedure $ZERO$. In the next subsection we show how to approximate $P_{\alpha,\ell}$, and in the subsection after that we show how to approximate $ZERO$.

3.2. Approximating $P_{\alpha,\ell}(x)$. In this subsection we show how to approximate $P_{\alpha,\ell}(x)$. The main tool in the approximation is the following lemma.

LEMMA 3.3. *For any polynomial $P(x)$, α and ℓ, such that $0 \le \alpha \le 2^\ell$, then*

$$P_{\alpha,\ell}(\omega_d^k) = \frac{1}{2^\ell} \sum_{m=0}^{2^\ell-1} P\left(\omega_d^{k+m\frac{d}{2^\ell}}\right) \omega_d^{-\alpha m \frac{d}{2^\ell}}$$

Proof. The proof is somewhat technical. We basically transform the right hand side to the left hand side. We start by substituting for $P(x)$ the expression $\sum_{j=0}^{d} a_j x^j$. This means that the right hand side is,

$$\frac{1}{2^\ell} \sum_{m=0}^{2^\ell-1} \sum_{j=0}^{d-1} a_j \omega_d^{j(k+m\frac{d}{2^\ell})} \omega_d^{-\alpha m \frac{d}{2^\ell}}$$

We can change the order of summation and rewrite it as,

$$\sum_{j=0}^{d-1} a_j \omega_d^{jk} \left[\frac{1}{2^\ell} \sum_{m=0}^{2^\ell-1} \left(\omega_d^{(j-\alpha)\frac{d}{2^\ell}} \right)^m \right]$$

Consider the expression between the brackets. If $j = \alpha \bmod 2^\ell$, then the expression is one, and otherwise it is equal to zero. Therefore we can rewrite it as,

$$\sum_{j:\, j \bmod 2^\ell = \alpha} a_j \omega_d^{jk} = P_{\alpha,\ell}(\omega_d^k).$$

□

We can use Lemma 3.3 to approximate $P_{\alpha,\ell}$. Define the following random variable,

$$A_m(\alpha,\ell,k) = \frac{1}{m}\sum_{i=1}^{m} P\left(\omega_d^{k+\beta_i\frac{d}{2^\ell}}\right)\omega_d^{-\alpha\beta_i\frac{d}{2^\ell}}$$

where β_i is chosen uniformly in the set $\{0,\dots,2^\ell-1\}$. The random variable $A_m(\alpha,\ell,k)$ is an approximation for $P_{\alpha,\ell}(\omega_d^k)$ using m sample points. The following lemma, which is an application of Chernoff bounds, states how "good" the approximation is.

LEMMA 3.4. *Let $P(x)$ be a polynomial such that $\max_j\{\|P(w_d^j)\|\} \le H$. There exists a constant c, such that for $m \ge cH^4\log\frac{2}{\delta}$ then, for any k,*

$$Prob\left[\|A_m(\alpha,\ell,k) - P_{\alpha,\ell}(\omega_d^k)\| \ge \frac{1}{16H}\right] \le \delta$$

Proof. The proof is an application of the Chernoff bounds, and the fact that $\|A_m(\alpha,\ell,k) - P_{\alpha,\ell}(\omega_d^k)\| \le 2H$. □

The main use of Lemma 3.4 would be through the following claim.

CLAIM 3.5. *Let a and b be complex numbers such that $\|a\| \le H$. If $\|a-b\| \le \frac{1}{16H}$ then,*

$$\left|\|a\|^2 - \|b\|^2\right| \le \frac{1}{4}.$$

3.3. Testing for Zero Polynomials. Our aim is to show that with high probability we can approximate each $ZERO(\alpha,\ell)$ correctly. This would imply that we can run the entire algorithm using the approximation, rather than testing $ZERO(\alpha,\ell)$. The approximating function is named $APPROX\_ZERO$. The main idea would be to approximate the power of the function. If the polynomial is the zero polynomial, then the power is zero, otherwise the power is at least one (since we assumed that the coefficients are integers).

For a polynomial $P(x) = \sum a_j x^j$, let $power(P) = \sum_j a_j^2$. In order to approximate the power of a polynomial we define the following random variable

$$B_{m_1,m_2}(\alpha,\ell) = \frac{1}{m_1}\sum_{i=1}^{m_1}\|A_{m_2}(\alpha,\ell,k_i)\|^2$$

where k_i is chosen uniformly in the set $\{0,\dots,d-1\}$. The random variable $B_{m_1,m_2}(\alpha,\ell)$ is an approximation of $power(P_{\alpha,\ell})$. The following lemma states how "good" the approximation is.

LEMMA 3.6. *Let $P(x)$ be a polynomial such that $H = \max_k\{\|P(w_d^k)\|\}$. There exists a constant c, such that for $m_1 \ge cH^4\log\frac{1}{\delta}$ and $m_2 \ge cH^4\log\frac{m_1}{\delta}$ then,*

$$Prob\left[|B_{m_1,m_2}(\alpha,\ell) - power(P_{\alpha,\ell})| \ge \frac{1}{2}\right] \le \delta$$

Proof. Lemma 3.3 shows how to compute $P_{\alpha,\ell}(\omega_d^k)$ from the values of $P(w_d^j)$. One can observe that the value of $P_{\alpha,\ell}(\omega_d^k)$ is a weighted average of the values of $P(w_d^j)$. From this we can deduce that $\max_k\{\|P_{\alpha,\ell}(\omega_d^k)\|\} \le H$.

The proof of the lemma uses Chernoff bounds. Since $\|P_{\alpha,\ell}(\omega_d^k)\|^2 \leq H^2$, by choosing m_1 values of k_i's at random, such that $m_1 \geq cH^4 \log\frac{1}{\delta}$, Theorem 2.1 guarantees that, with probability $1-\delta/2$,

$$\left| power(P_{\alpha,\ell}) - \frac{1}{m_1}\sum_{i=1}^{m_1} \|P_{\alpha,\ell}(\omega_d^{k_i})\|^2 \right| \leq \frac{1}{4}.$$

Based on Lemma 3.4, for every k_i, with probability $1-\delta/(2m_1)$, the random variable $A_{m_2}(\alpha,\ell,k_i)$ approximates $P_{\alpha,\ell}(\omega_d^{k_i})$ with error less than $1/16H$. By Claim 3.5 this implies that the difference between $\|A_{m_2}(\alpha,\ell,k_i)\|^2$ and $\|P_{\alpha,\ell}(\omega_d^{k_i})\|^2$ is at most $1/4$, which completes the proof of the lemma. □

Now we are ready to define the function $APPROX\_ZERO(\alpha,\ell)$. In order to compute $APPROX\_ZERO(\alpha,\ell)$, we first compute $B_{m_1,m_2}(\alpha,\ell) = b$, where m_1 and m_2 are chosen according to Lemma 3.6, then if b is smaller than $1/2$ we return $TRUE$, otherwise we return $FALSE$. The following theorem states that, with high probability, this is a good approximation.

THEOREM 3.7. *For any ℓ and α, $0 \leq \alpha \leq 2^\ell$,*

$$Prob[APPROX\_ZERO(\alpha,\ell) \neq ZERO(\alpha,\ell)] \leq \delta$$

Proof. It is easy to show that if $APPROX\_ZERO(\alpha,\ell) \neq ZERO(\alpha,\ell)$, then

$$|B_{m_1,m_2}(\alpha,\ell) - power(P_{\alpha,\ell})| \geq \frac{1}{2}$$

and the theorem follows from Lemma 3.6. □

3.4. Approximating a single coefficient. We need to approximate each coefficient that we found at the end of the algorithm with an error of at most ϵ. Given an exponent e, we show how to approximate its coefficients, a_e.

Define the random variable $COEF_m(e)$ to be,

$$COEF_m(e) = \left\lfloor \frac{1}{2} + \frac{1}{m}\sum_{i=1}^{m} P(\omega_d^{\beta_i})\omega_d^{-\beta_i e} \right\rfloor$$

where β_i is chosen uniformly from $\{0,\ldots,d-1\}$. The random variable $COEF_m(e)$ is an approximation of the coefficient a_e using m sample points. The following lemma states how close is the random variable $COEF_m(e)$ to the coefficient a_e.

LEMMA 3.8. *Let P be a polynomial and $H = \max_j\{\|P(\omega_d^j)\|\}$, then for $m \geq 32H^2\log\frac{2}{\delta}$,*

$$Prob\left[COEF_m(e) \neq a_e\right] \leq \delta$$

Proof. If $COEF_m(e) \neq a_e$ then $|COEF_m(e) - a_e| \geq 1$, since they are both integers. In such a case, $|\frac{1}{2} + \frac{1}{m}\sum_{i=1}^{m} P(\omega_d^{\beta_i})\omega_d^{-\beta_i e} - a_e| \geq 1/2$ The claim of the lemma follows from Chernoff bounds. □

3.5. Univariate Interpolation Algorithm: Small Coefficients. We can now use the building blocks in this section to show the following result.

THEOREM 3.9. *Let P be a non-zero univariate polynomial with at most t integer coefficients, each of magnitude at most L, and $H = \max_k\{\|P(w_d^k)\|\}$. The running time of the algorithm is $O(H^8 t \log d \log^2 \frac{1}{\delta'} + tH^2 \log \frac{1}{\delta}) = poly(n, t, \log d, \log L)$, where $\delta' = O(\frac{1}{H^8 t \log d \log^2 \frac{1}{\delta}})$*

Proof. There are $O(t \log d)$ recursive calls. Each time in $APPROX\_ZERO$ we test $O(H^4 \ln \frac{1}{\delta})$ different k_i's. For each k_i we use $O(H^4 \log \frac{1}{\delta})$ points to approximate it. At the end, for each of the t coefficients we call the procedure $COEF$ and use $O(H^2 \log \frac{1}{\delta})$ points. □

The main drawback of the above result is that the running time depends on H, which could be proportional to the largest coefficient, i.e. L, and not polynomial in the bit representation of the coefficients, i.e. $\log L$. In the next subsection we show how to overcome this deficiency.

4. Handling large coefficients. The aim of this section is to modify the algorithm of the previous section so that its running time would be polynomial in $\log L$. The technique we use in handling large coefficients is similar in spirit to the scaling techniques in graph algorithms. We first find the exponents whose coefficients are "very" large. We approximate the coefficients of those exponents and subtract them from the polynomial. Since we only approximate the coefficients, the new polynomial may still have for those exponents non-zero coefficients. The main advantage is that the new polynomial has, in the worst case, the same number of non-zero coefficients, while the maximum value of a coefficient is smaller.

Consider a t-sparse polynomial $P(x)$, i.e.

$$P(x) = \sum_{j=1}^{t} a_{e_j} x^{e_j}.$$

As before, let the parameter d be the smallest power of two larger than the degree, i.e. $d = 2^{\lceil \log(1+\max_j\{e_j\}) \rceil}$. Similarly, let L be the smallest power of two that is larger than the maximum coefficient in absolute value, i.e. $L = 2^{\lceil \log(\max_j\{|a_{e_j}|\}) \rceil}$.

Our aim, is to perform the interpolation in time that is polynomial in t, $\log d$ and $\log L$. The following is the basic idea of the algorithm. Given L, the bound on the largest coefficient, we consider the sets $Z_1 = \{e_j \ : \ L/2 \leq |a_{e_j}| \leq L\}$ and $Z_2 = \{e_j \ : \ \frac{L}{8\sqrt{t}} \leq |a_{e_j}| \leq L\}$. We find a set Y, such that $Z_1 \subset Y \subset Z_2$. Once we found the set Y (in time polynomial in t, $\log d$ and $\log L$), we approximate each of the coefficients of the exponents in Y with an absolute error of less than a $L/4$. This implies that, with high probability, the difference between the real coefficient and the approximated coefficient is less than $L/4$. For each $e_j \in Y$, denote by γ_{e_j} the approximated value of the coefficient a_{e_j} and in order to ensure that the new polynomial is integral we enforce γ_{e_j} be integral. More formally, let Q be the polynomial we created,

$$Q(x) = \sum_{e_j \in Y} \gamma_{e_j} x^{e_j}.$$

Consider the polynomial $P'(x) = P(x) - Q(x)$. Assuming that $Z_1 \subset Y \subset Z_2$, and $|\gamma_{e_j} - a_{e_j}| \leq L/4$, the polynomial P' is t-sparse, and has integer coefficients of size at most $L/2$. This implies that in polynomial time we reduced the problem of recovering the coefficients of a polynomial with maximum coefficient L (i.e. $P(x)$), to the problem of recovering the coefficients whose maximum coefficient is $L/2$ (i.e. $P'(x)$). Therefore after $O(\log L)$ such iterations the maximum coefficient is $O(\sqrt{t})$. Once the coefficients are of size $O(\sqrt{t})$ the algorithm of Theorem 3.9 would recover all the coefficients.

4.1. Approximating the power. The main idea is that with a sample size that depends on t and is independent of L we can approximate $power(P_{\alpha,\ell})$. We start by modifying the approximation of $P_{\alpha,\ell}$.

LEMMA 4.1. *Let P be a t-sparse polynomial each of whose coefficient is bounded in magnitude by L. There exists a constant c, such that if $m \geq ct^4 \log \frac{2}{\delta}$ then, for any k,*

$$Prob\left[\|A_m(\alpha, \ell, k) - P_{\alpha,\ell}(\omega_d^k)\| \geq \frac{L}{64t}\right] \leq \delta$$

As before, this implies the following about the difference in the magnitude.

CLAIM 4.2. *Let a and b be complex numbers such that $\|a\| \leq tL$. If $\|a - b\| \leq \frac{L}{64t}$ then,*

$$\Big|\|a\|^2 - \|b\|^2\Big| \leq \frac{L^2}{16}.$$

We can now claim the result about approximating the power.

LEMMA 4.3. *Let P be a t-sparse polynomial each of whose coefficient is bounded in magnitude by L. There exists a constant c, such that if $m_1 \geq ct^4 \log \frac{1}{\delta}$ and $m_2 \geq ct^4 \log \frac{m_1}{\delta}$, then*

$$Prob\left[|B_{m_1,m_2}(\alpha, \ell) - power(P_{\alpha,\ell})| \leq \frac{3L^2}{32}\right] \leq \delta$$

Proof. The polynomial P is t-sparse, furthermore, each coefficient is bounded by L. Therefore, the magnitude of $P(w_d^k)$ is bounded by tL, and the magnitude of $P_{\alpha,\ell}(w_d^k)$ is bounded by tL. Once we bounded the magnitude of $P_{\alpha,\ell}$ we can use the Chernoff bounds to claim that with probability $1 - \delta/2$ the difference between $power(P_{\alpha,\ell})$ and $\frac{1}{m_1}\sum_{1\leq j\leq m_1} \|P_{\alpha,\ell}(\omega_d^{k_j})\|^2$, is less than $L^2/32$.

Based on Lemma 4.1, for every k_i, with probability $1-\delta/(2m_1)$ the random variable $A_{m_2}(\alpha, \ell, k_i)$ approximates $P_{\alpha,\ell}(\omega_d^{k_i})$ with error less than $L/(64t)$. By Claim 4.2 this implies that the difference between $\|A_{m_2}(\alpha, \ell, k_i)\|^2$ and $\|P_{\alpha,\ell}(\omega_d^{k_i})\|^2$ is at most $L^2/16$. □

Now we define the equivalent of the $ZERO$ and $APPROX\_ZERO$ procedures. Let $LARGE(P, L)$, where P is a polynomial and L an integer, be $TRUE$ if and only if P has some coefficient a_j such that $|a_j| \geq L/2$. As before, we cannot compute $LARGE(P, L)$ directly, but we can approximate it. We approximate the value of $LARGE(P_{\alpha,\ell}, L)$ by

$APPROX\_LARGE(\alpha, \ell, L)$ which is defined as follows: Compute $B_{m_1,m_2}(\alpha, \ell) = b$ and if $b \geq \frac{5L^2}{32}$ return $TRUE$, otherwise return $FALSE$.

LEMMA 4.4. *Let P be a t-sparse polynomial each of whose coefficient is bounded in magnitude by L. For any α and ℓ, $0 \leq \alpha \leq 2^\ell$, then*

$$Prob[APPROX\_LARGE(\alpha, \ell, L) = FALSE \textbf{ and } LARGE(P_{\alpha,\ell}, L) = TRUE] \leq \delta$$

Proof. Assume that $LARGE(P_{\alpha,\ell}, L)$ is $TRUE$. This implies that $power(P_{\alpha,\ell}) \geq L^2/4$, since some coefficient is larger than $L/2$. With probability $1 - \delta$

$$|B_{m_1,m_2}(\alpha, \ell) - power(P_{\alpha,\ell})| \geq \frac{3L^2}{32}.$$

This implies that with probability $1-\delta$ we have $B_{m_1,m_2}(\alpha, \ell) \geq \frac{5L^2}{32}$, which is equivalent to
$APPROX\_LARGE(\alpha, \ell, L)$ being $TRUE$. □

LEMMA 4.5. *Let P be a t-sparse polynomial each of whose coefficient is bounded in magnitude by L. For any α and ℓ, $0 \leq \alpha \leq 2^\ell$, then*

$$Prob[APPROX\_LARGE(\alpha, \ell, L) = TRUE \textbf{ and } LARGE(P_{\alpha,\ell}, \frac{L}{4\sqrt{t}}) = FALSE] \leq \delta$$

Proof. Assume that $LARGE(P_{\alpha,\ell}, \frac{L}{4\sqrt{t}})$ is $FALSE$. This implies that $power(P_{\alpha,\ell}) \leq L^2/64$, since no coefficient is larger than $\frac{L}{8\sqrt{t}}$. With probability $1 - \delta$

$$|B_{m_1,m_2}(\alpha, \ell) - power(P_{\alpha,\ell})| \geq \frac{3L^2}{32}.$$

This implies that with probability at most δ we have $B_{m_1,m_2}(\alpha, \ell) \geq \frac{5L^2}{32}$, which is equivalent to $APPROX\_LARGE(\alpha, \ell, L) = TRUE$. □

The above two lemmas show that if $APPROX\_LARGE(\alpha, \ell, L)$ is $TRUE$, then, with high probability, $P_{\alpha,\ell}$ is a non-zero polynomial (and even has a coefficient larger than $\frac{L}{8\sqrt{t}}$), and if $P_{\alpha,\ell}$ has a coefficient larger than $L/2$, then with high probability $APPROX\_LARGE(\alpha, \ell, L)$ is $TRUE$.

Note that the number of recursive calls when we use the procedure $LARGE$ is bounded, as before, by $t \log d$, since every time $LARGE$ is $TRUE$, we are focusing on at least one of the t coefficients.

4.2. Approximating a single coefficient. The approximation of a single coefficient is done as in the case of small coefficients, in Section 3.4. The only difference is that we need to approximate each coefficient that we found with absolute error less than $L/4$.

LEMMA 4.6. *There is a constant c, such that for $m \geq ct^2 \log \frac{1}{\delta}$,*

$$Prob\left[|COEF_m(e) - a_e| \geq L/4\right] \leq \delta.$$

4.3. Putting it all together. The basic building block would be a procedure $FIND(P, L)$ that receives a polynomial P with coefficients of magnitude at most L, and returns a list of exponents and their respective coefficients such that with high probability this list includes all the coefficients larger than $L/2$ and no coefficient smaller than $\frac{L}{8\sqrt{t}}$.

The procedure $FIND(P, L)$ works as the algorithm in Section 3, but uses the subroutine $APPROX\_LARGE$ instead of $APPROX\_ZERO$. At the beginning we start with a set Y_0 that has only 0. At phase ℓ, for each element $\alpha \in Y_\ell$, we create $\alpha_0 = \alpha$ and $\alpha_1 = \alpha + 2^\ell$. We add α_σ to $Y_{\ell+1}$ only if $APPROX\_LARGE(\alpha_\sigma, \ell + 1, L)$ is $TRUE$.

Since we are modifying the original polynomial we describe how to compute the output that the black box returns to the algorithm. We have a polynomial $R(x)$ which is initially identically zero. (This polynomial would be used to "change" the original polynomial P that the algorithm is approximating.) Each time the algorithm queries the black box on a point ω_d^k, we query the original black box on input (d, k) and get a reply $\rho(d, k)$ (which is simply $P(\omega_d^k)$ with finite accuracy), we reply to the algorithm with $\rho(d, k) - R(\omega_d^k)$.

The main algorithm runs in phases. We maintain that after the i-th phase, with high probability, $P(x) - R(x)$ is a polynomial with at most t non-zero coefficients and its maximum coefficient is at most $L/2^i$. The i-th phase consists from the following steps: (1) We call $FIND(P - R, L/2^{i-1})$ that returns a set Y_i. With high probability the set Y_i includes all the coefficients in the range $L/2^i$ to $L/2^{i-1}$. (2) For each exponent in $e \in Y_i$ we approximate its coefficient a_e by γ_e, such that the difference is at most $L/2^{i+1}$. Let $Q_i(x) = \sum_{e \in Y_i} \gamma_e x^e$. (3) We add the polynomial $Q_i(x)$ to the polynomial $R(x)$.

Once the size of the coefficients is $O(\sqrt{t})$ the algorithm uses the algorithm for small coefficients (see Theorem 3.9) to completely recovers the coefficients, and adds this polynomial to R. At this point, with high probability, the polynomial $R(x)$ at is identical to the polynomial $P(x)$.

We have established the following theorem.

THEOREM 4.7. *There exists a randomized algorithm that interpolates a t-sparse polynomial, whose coefficients are integers less than L. The algorithm runs in time polynomial in t, $\log d$ and $\log L$.*

5. Precision accuracy. Our algorithms use complex numbers. When we measure the bit complexity we need to specify the accuracy to which we require the complex numbers to be. This in turn also determines the bit complexity of the algorithm, since it determines the size of the numbers that the algorithm operates on. Since most of our operations are simply averaging over a set of random inputs, the algorithm requires only a low accuracy. The following claim shows how we can bound the precision accuracy.

CLAIM 5.1. *Adding m numbers with precision accuracy ϵ/m results in an output that has an absolute error of at most ϵ. Furthermore, if each of the m numbers is bounded in absolute value by H then their description requires only $O(\log H + \log \frac{m}{\epsilon})$ bits to guarantee a precision accuracy of ϵ in the result.*

The values used in $COEF_m(e)$ and $A_m(\alpha, \ell, k)$ are bounded by tL. This implies that using $O(\log tL + \log \frac{m}{\epsilon})$ bits (and precision accuracy ϵ/m) in their computation, we introduce an additive error of at most $O(\epsilon)$. Since those random variables are used only as an approximation, the additional error could be absorbed in the error margins that we assumed before. (Recall that $B_{m_1,m_2}(\alpha, \ell)$ is define through A_m, and therefore the precision accuracy of B_{m_1,m_2} is determine by the precision accuracy of A_m.)

6. Approximation by polynomials. Assume that rather than recovering all the coefficients of the polynomial we are interested in recovering all the "large" coefficients of the polynomial. This is a very interesting case, since in many cases we may regard the other coefficients as "noise" that is not interesting.

The formal scenario is the following. As before we are given a polynomial P as a black box, however, we are not guaranteed that it is either t-sparse or has integer coefficients. We assume that there is some polynomial Q, with t integer coefficients, such that the power of $P - Q$ is at most ν.

Our algorithm will recover a polynomial Q', such that the power of $P - Q'$ is bounded by $\nu + O(\epsilon)$. The number of recursive calls done by the algorithm is bounded by $O((t + \nu)\log d)$. The correctness is argued as before (each coefficient larger than one would be reached, with high probability). The running time can be bounded by observing that the difference between P and Q can cause the algorithm, for each specific value of ℓ, to search at most $O(\nu)$ values that would not lead to a coefficient of size at least one. The time to perform a single $APPROX\_ZERO$ or $APPROX\_LARGE$ remains the same.

We made a restriction that the approximating polynomial Q has integer coefficients. This restriction can be easily removed in the following way. Assume that there is a polynomial R that has real coefficients, and the power of $P - R$ is at most ν. Consider the polynomial R' that is achieved by multiplying the coefficients of R by t/ϵ, and rounding them to the nearest integer. The difference between $R(x)$ and $R'(x) * (\epsilon/t)$ is bounded by ϵ. This implies that the polynomial $P(x) * (t/\epsilon)$ has a good approximation by an integer polynomial R'.

7. Multi-variate polynomials. The technique that we developed for univariate polynomials extends to multi-variate polynomials, and uses ideas similar to [GL89, KM91] and [Zip79]. In this Section we give a sketch of the ideas of the extension.

The idea for multi-variate polynomials is the following. As before, We first recover all the "large coefficients". We proceed variable by variable, and find the exponents of the "large coefficients". Then we approximate the coefficients, and subtract them from the original polynomial. The main difference is that we recover the exponents of the interesting coefficients sequentially.

In the rest of this subsection we set the required definitions for the algorithm, and the test function that is used. Consider the polynomial,

$$P(x_1, \ldots, x_n) = \sum_{j=1}^{t} a_{e_{1,j},\ldots,e_{n,j}} x_1^{e_{1,j}} x_2^{e_{2,j}} \cdots x_n^{e_{n,j}}$$

As in the case of a single variable, we can compute a single coefficient as follows,

$$a_{e_1,\ldots,e_n} = \frac{1}{d^n}\sum_{k_1=0}^{d-1}\cdots\sum_{k_n=0}^{d-1} P(\omega_d^{k_1},\omega_d^{k_2},\cdots,\omega_d^{k_n})\omega_d^{-k_1e_1}\cdots\omega_d^{-k_ne_n}$$

Also, the Parseval identity holds,

$$\sum_{j=1}^{t} a^2_{e_{1,j},\ldots,e_{n,j}} = \frac{1}{d^n}\sum_{k_1=0}^{d-1}\cdots\sum_{k_n=0}^{d-1}\|P(\omega_d^{k_1},\omega_d^{k_2},\cdots,\omega_d^{k_n})\|^2$$

The polynomial $P_{\alpha,\ell}$ is replaced by $P_{<e_1,\ldots,e_{j-1}>,\alpha,\ell}$, which is the polynomial that includes only the coefficients in which the exponents of $x_1,\ldots,x_{j-1}$ are $e_1,\ldots e_{j-1}$ respectively, the power of x_j equals to α modulus 2^ℓ, and the exponents of the variables $x_{j+1},\ldots,x_n$ are unrestricted. The idea is that we are executing the univariate polynomial algorithm on the j-th input. The input $x_1,\ldots x_{j-1}$ can appear with only a specific power, while there is no restriction on the exponents of $x_{j+1},\ldots,x_n$. Formally,

$$P_{<e_1,\ldots,e_{j-1}>,\alpha,\ell}(x_1,\cdots,x_n) = x_1^{e_1}\cdots x_{j-1}^{e_{j-1}}\sum_{e_j \bmod 2^\ell=\alpha} x_j^{e_j}\sum_{e_{j+1}=0}^{d-1}\cdots\sum_{e_n=0}^{d-1} a_{e_1,\cdots,e_n}x_{j+1}^{e_{j+1}}\cdots x_n^{e_n}$$

We define POWER as follows,

$$POWER[e_1,\ldots,e_{j-1},\alpha,\ell] = E_{\vec{k}}[\|P_{<e_1,\ldots,e_{j-1}>,\alpha,\ell}(\vec{\omega}_d^{\vec{k}})\|^2]$$

Where $\vec{k}=k_1,\ldots,k_n$ and $k_j\in\{0,\ldots,d-1\}$, and $\vec{\omega}_d^{\vec{k}}$ is the input $x_1=\omega_d^{k_1},\cdots,x_n=\omega_d^{k_n}$. The expectation is uniform over all $\vec{k}$. As before, $POWER[e_1,\ldots,e_{j-1},\alpha,\ell]$ equals the sum of the squares of coefficients in the polynomial $P_{<e_1,\ldots,e_{j-1}>,\alpha,\ell}(x_1,\cdots,x_n)$.

We search the j-th variable before we continue to the $j+1$ variable. We find all the exponents prefixes of all the non-zero coefficients, upto the j-th component (there are at most t). With each such prefix we continue to search for the $j+1$th component. The argument for correctness is similar to the argument for the univariate polynomials.

Consider the following example: $P(x_1,x_2)=q_1(x_1)+q_2(x_1)x_2$, where q_1 and q_2 are polynomials. Consider the expression

$$\begin{aligned}E_{k_2}\left[\left(q_1(\omega_d^k)+q_2(\omega_d^k)\omega_d^{k_2}\right)\left(q_1(\omega_d^{-k})+q_2(\omega_d^{-k})\omega_d^{-k_2}\right)\right] &= E_{k_2}[q_1(\omega_d^k)q_1(\omega_d^{-k})]\\ &+ E_{k_2}[q_2(\omega_d^k)q_2(\omega_d^{-k})]\\ &+ E_{k_2}[q_1(\omega_d^k)q_2(\omega_d^{-k})\omega_d^{-k_2}]\\ &+ E_{k_2}[q_2(\omega_d^k)q_1(\omega_d^{-k})\omega_d^{k_2}]\end{aligned}$$

Note that the first two terms do not depend on x_2 while the last two terms evaluate to zero, due to the averaging over x_2. In general the averaging over all possible values of a certain variable causes the cross terms in the expression of the power to cancel out, therefore it has the effect of essentially "ignoring" this variable.

Acknowledgments. I would like to thank László Babai, Persi Diaconis, Marek Karpinski, Eyal Kushilevitz, Michael Rabin and Prasoon Tiwari, for the discussions that I had with each of them.

REFERENCES

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the* 20^{th} *Annual ACM Symposium on Theory of Computing*, pages 301–309, May 1988.

[GK87] D. Y. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in nc. In 28^{th} *Annual Symposium on Foundations of Computer Science, Los Angeles, California*, pages 166–172, October 1987.

[GKS90a] D. Y. Grigoriev, M. Karpinski, and M. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, 19(6):1059–1063, December 1990.

[GKS90b] D. Y. Grigoriev, M. Karpinski, and M. Singer. Interpolation of sparse rational functions without knowing bounds on exponents. In 31^{th} *Annual Symposium on Foundations of Computer Science, St. Louis*, pages 840–846, October 1990.

[GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 25–32. ACM, 1989.

[HR89] Torben Hagerup and Christine Rub. A guided tour to chernoff bounds. *Info. Proc. Lett.*, 33:305–308, 1989.

[Kar89] M. Karpinski. Boolean circuit complexity of algebraic interpolation problems. Technical Report TR-89-027, ICSI, 1989.

[KM91] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. In *Proceedings of STOC '91*, pages 455–464, may 1991.

[Zip79] R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, pages 216–226. Springer Lecture notes in computer science, vol. 72, 1979.