Saarland University  
Faculty of Natural Sciences and Technology I  
Department of Computer Science  
Master’s Program in Computer Science

Master’s Thesis

**Randomness Efficient Testing of Sparse Black Box Polynomials and Related Tests**

submitted by

**Gorav Jindal**

on December 12, 2013

Supervisor

Prof. Dr. Markus Bläser

Reviewers

Prof. Dr. Markus Bläser  
Prof. Dr. Raimund Seidel
Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, December 12, 2013

Gorav Jindal
I would like to thank Prof. Markus Bläser, my adviser, for introducing me to these interesting problems and also for the helpful advice. I would also like to extend my sincere gratitude to Prof. Raimund Seidel for reviewing this thesis.

I thank Thatchaphol Saranurak who helped me refine several arguments.

I also thank my parents for the encouragement.
## Contents

1. Polynomial Identity Testing .......................... 1
   1.1 Introduction ...................................... 1
   1.2 Our Results ...................................... 2
   1.3 Preliminaries ...................................... 3
   1.4 Algorithms for PITRBB and PITQBB ................. 5
      1.4.1 Improved deterministic algorithm ............... 5
      1.4.2 Randomized Algorithm when the degree is \text{poly}(m) ... 7
      1.4.3 Reducing Random bits ........................... 10

2. Polynomial Interpolation .............................. 15
   2.1 Introduction ....................................... 15
   2.2 New Black-box Model .............................. 16
   2.3 Our approach ...................................... 17
   2.4 Preliminaries ...................................... 18
   2.5 Univariate interpolation .......................... 20
      2.5.1 Finding a \textit{Good} prime .................... 20
      2.5.2 Finding many \textit{Good} primes ................ 23
      2.5.3 Main Algorithm for Interpolation ............... 24
   2.6 Multivariate Interpolation ......................... 27
   2.7 Conclusion and applications ....................... 28
Chapter 1

Polynomial Identity Testing

1.1 Introduction

Polynomial identity testing is the problem of testing if a polynomial $P$ is equal to zero. It has always been one of the most important problems in theoretical computer science. Many famous problems reduce to polynomial identity testing. Agrawal and Biswas [AB03] showed that testing whether a number $n$ is prime or not reduces to testing whether following polynomial $P(x)$ is zero or not $\mod n$.

$$P(x) = (1 + x)^n - 1 - x^n$$

It was shown in [AB03] that number $n$ is prime if and only if $P(x) = 0 \pmod n$. Agrawal, Kayal and Saxena [AKS04] were able to derandomize identity testing of $P(x)$ to get the first polynomial time deterministic algorithm for primality testing. It was observed in [MVV87] that the determinant polynomial of Tutte matrix of a graph is non-zero if and only if the graph has a perfect matching.

Polynomial identity testing has been studied in various models. Complexity of this problem is highly dependent on the model in which we study it. For example, if polynomial is given as a list of coefficients and monomials then this problem is trivial. Arithmetic circuits and black-box model have been two of the most prominent models in which polynomial identity testing has been studied extensively.

Schwartz [Sch80] and Zippel [Zip79] devised randomized algorithm for polynomial identity testing. They observed that if we evaluate the polynomial at a random point from a large enough domain then we get a non-zero answer with high probability, if the input polynomial was non-zero. The size of domain depends upon the degree of the input polynomial. Their observation is famously known as the Schwartz-Zippel Lemma and can be found in almost any literature dealing with algebraic computations, such as [GG03].
CHAPTER 1. POLYNOMIAL IDENTITY TESTING

Chen and Kao [CK97] introduced a new approach to polynomial identity testing over black-box model. They used rational approximation of smartly chosen irrational numbers as the evaluation point. They were able to show that we get a non-zero answer for this point with high probability if the input polynomial was non-zero. Algorithm by Chen and Kao [CK97] used only \( \sum_{i=1}^{n} \lceil \log(d_i + 1) \rceil \) random bits, where \( n \) is the number of variables and \( d_i \) is the bound on the degree of the \( i \)-th variable. But their algorithm only worked for polynomials over integers. Lewin and Vadhan [LV98] extended the idea of Chen and Kao to get a randomized algorithm using \( \sum_{i=1}^{n} \lceil \log(d_i + 1) \rceil \) random bits over any field. Whereas Chen and Kao [CK97] used square roots of prime numbers to construct the irrational numbers, Lewin and Vadhan [LV98] used square roots of irreducible polynomials over the underlying field. Lewin and Vadhan [LV98] also showed almost matching lower bound on the number of random bits used. More precisely, they showed a lower bound of \( (1 - o(1)) \sum_{i=1}^{n} \log(d_i + 1) \) random bits for any algorithm which made only \( \text{poly}(n) \) queries to black-box. This approach also gave a new time-error trade-off instead of regular randomness-error trade-off.

Agrawal and Biswas [AB03] introduced yet another framework for polynomial identity testing using the same number of random bits, i.e., \( \sum_{i=1}^{n} \log(d_i + 1) \). They constructed a set of polynomials such that any non-zero polynomial cannot be divisible by too many of the polynomials in the set of the polynomials they constructed.

Goal of all the approaches to polynomial identity testing has been to get a deterministic algorithm for this problem. Namely, given an arithmetic circuit \( C \) of input length \( l \), can we find a deterministic algorithm which runs in time \( \text{poly}(l) \) and tests whether the polynomial represented by \( C \) is zero? Currently this problem is known to be in complexity class \( \text{coRP} \). It is a long standing open problem to show whether this problem belongs to complexity class \( P \).

Reducing the number of random bits has been an intermediate goal to understand this problem better and perhaps solve it in the long run. We also study the problem of reducing random bits for black-box model over reals.

1.2 Our Results

We study the polynomial identity testing in the black-box model. We consider sparse polynomials over the reals. By sparse, we mean that we are guaranteed that polynomial has at most \( m \) monomials for some given number \( m \). More specifically, we study the polynomial identity testing in the model which can be pictured as follows.
CHAPTER 1. POLYNOMIAL IDENTITY TESTING

\[ P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} c_i \prod_{j=1}^{n} x_j^{\alpha_{ij}}. \]

We say that the degree of \( P(x_1, x_2, \ldots, x_n) \) is bounded by \( d \) if

\[ \forall i \in [m], \forall j \in [n] \alpha_{ij} \leq d. \]

Here we have that for \( k \neq l : (\alpha_{k1}, \alpha_{k2}, \ldots, \alpha_{kn}) \neq (\alpha_{l1}, \alpha_{l2}, \ldots, \alpha_{ln}). \)

As mentioned earlier, we will usually work over polynomials over the field of reals. Hence \( c_i \in \mathbb{R} \) in the above definition.

Now we define the problem which we want to solve.

1.3 Preliminaries

In this section, we formally define the problem we want to solve and corresponding notations.

**Definition 1.1.** A polynomial \( P(x_1, x_2, \ldots, x_n) \) is called \( m \)-sparse if it can be written in the following form

\[ P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} c_i \prod_{j=1}^{n} x_j^{\alpha_{ij}}. \]

The black-box model is used to evaluate the polynomial. The black-box returns whether the polynomial is zero or not on this point. We want both deterministic and randomized algorithm for this problem.

This problem was extensively studied in [BE11]. Bläser and Engels [BE11] devised a deterministic algorithm which runs in time \( \tilde{O}(m^3n^3) \). They also devised a randomized algorithm which runs in time \( \text{poly}(n, \log m) \) and uses \( O(\log^2 m) \) random bits. It is not difficult to show that any randomized algorithm which runs in time \( \text{poly}(n, \log m) \) has to use at least \( \Omega(\log m) \) random bits. We present a new deterministic algorithm which runs in time \( \tilde{O}(m^2n) \). For the polynomials whose degree is bounded by \( \text{poly}(m) \) and whose coefficients are rationals, we present a randomized algorithm which runs in time \( \text{poly}(n, \log m) \) using only \( O(\log m) \) random bits. Also, we tweak the randomized algorithm in [BE11] to reduce the number of random bits to \( O\left(\frac{\log^2 m}{\log \log m}\right) \).
CHAPTER 1. POLYNOMIAL IDENTITY TESTING

Problem 1.2 (PITRBB). Given an $m$-sparse polynomial $P(x_1, x_2, \ldots, x_n) \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ as a black-box, PITRBB is the problem of testing whether $P(x_1, x_2, \ldots, x_n)$ is the zero polynomial.

Problem 1.3 (PITQBB). Given an $m$-sparse polynomial $P(x_1, x_2, \ldots, x_n) \in \mathbb{Q}[x_1, x_2, \ldots, x_n]$ as a black-box, PITQBB is the problem of testing whether $P(x_1, x_2, \ldots, x_n)$ is the zero polynomial.

We use the notation $\tilde{O}(t)$ to denote the complexity $O(t \cdot (\log t)^{O(1)})$. Whenever we say “time” in the discussion that follows, we will always mean number of bit operations. We shall use the following basic fact about prime numbers which follows from the famous Prime Number Theorem.

Fact 1.4. The $n$-th prime number’s magnitude is of order $\Theta(n \log n)$.

Fact 1.5. Two positive numbers of bit length $m$ and $n$ respectively can be multiplied using $O(mn)$ number of bit operations.

Corollary 1.6. If we have $t$ positive numbers $n_1, n_2, \ldots, n_t$ of bit length $k$ each then their multiplication, i.e., $n_1 \cdot n_2 \cdot \ldots \cdot n_t$ can be computed using $O((kt)^2)$ number of bit operations.

Corollary 1.7. Given a positive number $M$ of bit length $m$ and a positive number $n$, $M^n$ can be computed using $O((mn)^2)$ number of bit operations.

Lemma 1.8 (THEOREM 5.4.1 in [BS96]). For positive integers $a, e$ and $n \geq a$, there is an algorithm to compute $a^n \mod n$ in $O((\log e)(\log n)^2)$ bit operations.

Lemma 1.9 (Lemma 4.4 in [BE11]). Let $N \geq 1$ be a positive number. We can find a prime number $p$ satisfying $N < p \leq 2N$ with success probability $\geq 1 - \beta$, using $O(\log N + \log \frac{1}{\beta})$ random bits and $\text{poly}(\log N, \log \frac{1}{\beta})$ bit operations.

We shall need Descartes’s Rule of Signs in the ensuing discussion. A proof of Descartes’s Rule of Signs can be found in [Wan04].

Theorem 1.10 (Descartes’s Rule of Signs). Let $p(x) = a_1x^{b_1} + a_2x^{b_2} + \ldots + a_nx^{b_n}$ be a polynomial with nonzero real coefficients $a_i$, where the $b_i$ are integers satisfying $0 \leq b_1 < b_2 < \ldots < b_n$. Then the number of positive real zeros of $p(x)$ (counted with multiplicities) is either equal to the number of variations in sign in the sequence $a_1, a_2, \ldots, a_n$ of the coefficients or less than that by an even whole number.

Corollary 1.11. Let $p(x) = a_1x^{b_1} + a_2x^{b_2} + \ldots + a_nx^{b_n}$ be a polynomial with nonzero real coefficients $a_i$, where the $b_i$ are integers satisfying $0 \leq b_1 < b_2 < \ldots < b_n$. Then the number of positive real zeros of $p(x)$ (counted with multiplicities) is less than $n$. 
We shall also need the following trivial lemma for bounding probabilities of error in the analysis of our randomized algorithm.

**Lemma 1.12.** Let $A$ and $B$ be two probabilistic events such that $\Pr[A \mid B] \geq 1 - \epsilon$ and $\Pr[B] \geq 1 - \delta$ for some non-negative real numbers $\delta$ and $\epsilon$. Then $\Pr[A] \geq 1 - (\delta + \epsilon)$.

**Proof.** By the law of total probability, we have

\[
\Pr[A] = \Pr[A \mid B] \cdot \Pr[B] + \Pr[A \mid \neg B] \cdot \Pr[\neg B] \\
\geq \Pr[A \mid B] \cdot \Pr[B] \\
\geq (1 - \epsilon)(1 - \delta) \\
= 1 - (\delta + \epsilon) + \epsilon \cdot \delta \\
\geq 1 - (\delta + \epsilon).
\]

\[\square\]

### 1.4 Algorithms for PITRBB and PITQBB

Now we present an improved deterministic algorithm. This algorithm is faster than the deterministic algorithm presented in [BE11]. For PITRBB, we usually want randomized algorithms running in time $\text{poly}(n, \log m)$. We present two randomized algorithms. The first randomized algorithm achieves a matching upper bound of $O(\log m)$ random bits for a special case of PITQBB. In this special case, we need that the degree of the polynomial is bounded by $\text{poly}(m)$. Second randomized algorithm is for PITRBB. It uses only $O(\frac{\log^2 m}{\log \log m})$ random bits.

#### 1.4.1 Improved deterministic algorithm

As mentioned earlier, Bläser and Engels [BE11] devised a deterministic algorithm for PITRBB which runs in time $\tilde{O}(m^3 n^3)$. Their algorithm relies on a lemma which is similar to Schwartz-Zippel, but for sparse polynomials. We use a completely different approach here to get an improved deterministic algorithm for PITRBB. This approach dates back to [BOSS]. Ben-or and Tiwari [BOSS] used this approach to devise the first deterministic algorithm for interpolating sparse multivariate polynomials over the reals. But we can also use this approach for polynomial identity testing.

**Lemma 1.13.** Let $p_i$ be the $i$-th prime number. Let $P_k = (p_1^{k-1}, p_2^{k-1}, \ldots, p_n^{k-1})$ for $1 \leq k \leq m$. Then any non-zero $m$-sparse polynomial is zero on at most $m - 1$ of the points $P_1, P_2, \ldots, P_m$. 
Proof. Let \( P(x_1, x_2, \ldots, x_n) \) be a non-zero \( m \)-sparse polynomial. Assume that \( P(x_1, x_2, \ldots, x_n) \) evaluates to zero all \( m \) points \( P_1, P_2, \ldots, P_m \). Let

\[
P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} c_i \prod_{j=1}^{n} x_j^{\alpha_{ij}}
\]

Then

\[
P(P_k) = P(p_1^{k-1}, p_2^{k-1}, \ldots, p_n^{k-1}) = \sum_{i=1}^{m} c_i \prod_{j=1}^{n} (p_j^{k-1})^{\alpha_{ij}}
\]

\[
= \sum_{i=1}^{m} c_i \left( \prod_{j=1}^{n} p_j^{\alpha_{ij}} \right)^{k-1}
\]

Let us use \( q_i \) to denote the integer \( \prod_{j=1}^{n} p_j^{\alpha_{ij}} \). Since, for \( k \neq l \): \((\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in}) \neq (\alpha_{l1}, \alpha_{l2}, \ldots, \alpha_{ln})\).

We get that \( q_k \neq q_l \) for \( k \neq l \). Evaluation of \( P \) at points \( P_1, P_2, \ldots, P_m \) can be represented as the following linear system.

\[
\begin{pmatrix}
1 & q_1 & \cdots & q_1^{m-1} \\
1 & q_2 & \cdots & q_2^{m-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & q_m & \cdots & q_m^{m-1}
\end{pmatrix}
\begin{pmatrix}
c_1 \\
c_2 \\
\vdots \\
c_m
\end{pmatrix}
= 
\begin{pmatrix}
P(P_1) \\
P(P_2) \\
\vdots \\
P(P_m)
\end{pmatrix}
\]

Above system is of the form \( A^T c = r \). Here \( A \) is Vandermonde matrix. \( A \) is known to be invertible when all \( q_i \)'s are pairwise distinct, hence \( A^T \) is also invertible. Since \( P \) is non-zero, at least one of the \( c_i \)'s is non-zero. Hence \( c \neq 0 \).

By assumption, \( P(x_1, x_2, \ldots, x_n) \) evaluates to zero on all points \( P_1, P_2, \ldots, P_m \), hence \( r = 0 \). Therefore we have \( A^T c = 0 \). But this means kernel of \( A^T \) contains a non-zero vector \( c \), making \( A^T \) singular. But \( A^T \) is invertible. Hence \( P(x_1, x_2, \ldots, x_n) \) cannot evaluate to zero on all \( m \) points \( P_1, P_2, \ldots, P_m \).

With the previous lemma, we are ready to describe our deterministic algorithm for PITRBB.
Algorithm 1.1 Deterministic Algorithm for PITRBB

**Input:** A black-box for an $m$-sparse multivariate polynomial $P(x_1, x_2, \ldots, x_n)$.

**Output:** Determine whether $P(x_1, x_2, \ldots, x_n)$ is zero or not.

1. Determine the first $n$ primes $p_1, p_2, \ldots, p_n$.
2. For $k = 1$ to $m$
   a. Query the black-box at the point $P_k = (p_1^{k-1}, p_2^{k-1}, \ldots, p_n^{k-1})$.
3. If all the answers were zero in Step 2a then output “$P(x_1, x_2, \ldots, x_n)$ is zero” else output “$P(x_1, x_2, \ldots, x_n)$ is non-zero”.

Theorem 1.14. Algorithm 1.1 solves PITRBB in time $\tilde{O}(m^2 n)$.

**Proof.** Correctness of Algorithm 1.1 is a trivial consequence of Lemma 1.13. For finding first $n$ primes $p_1, p_2, \ldots, p_n$ in Step 1 we just need to iterate on first $\Theta(n \log n)$ numbers and checks for primes. Using AKS algorithm \[AKS04\], we can test primality of any number of order $\Theta(n \log n)$ in time $O((\log n)^{O(1)})$. Hence Step 1 takes $\tilde{O}(n)$ time. We can compute each $P_{k+1}$ from $P_k$. Computing $P_2$ takes $O(n)$ time as Step 2 takes $\tilde{O}(n)$ time. To compute $i$-th component of $P_{k+1}$ from $P_k$, we need to multiply $p_i$ with $p_i^{k-1}$. This multiplication takes $O(k \log^2 n)$ due to Fact 1.5 and the fact that $p_i^{k-1}$ has size of $O(k \log n)$ bits. Hence $P_{k+1}$ can be computed in $O(kn \log^2 n) \in \tilde{O}(kn)$ time. Computing $P_1, P_2, \ldots, P_m$ takes $\sum_{k=1}^{m} \tilde{O}(kn)$ time, which is $\tilde{O}(m^2 n)$. Hence Step 2 takes $\tilde{O}(m^2 n)$ time in total. Thus Algorithm 1.1 runs in the time $\tilde{O}(m^2 n)$.

1.4.2 Randomized Algorithm when the degree is $\text{poly}(m)$

We assume that $n \leq m$. If $n > m$ then Algorithm 1.1 runs in time $\tilde{O}(m^2 n) = \tilde{O}(n^3)$ time. Hence if $n > m$, we already have a deterministic algorithm for PITRBB running in time $\text{poly}(n)$. As described earlier, any randomized algorithm which runs in time $\text{poly}(n, \log m)$ has to use at least $\Omega(\log m)$ random bits to solve PITRBB. Ideally, we want to solve PITQBB for unbounded degree polynomials in time $\text{poly}(n, \log m)$ using only $O(\log m)$ random bits. This would match the known lower bound of $\Omega(\log m)$ random bits. In this subsection, we describe the randomized algorithm which runs in time $\text{poly}(n, \log m)$ and solves PITQBB (and not PITRBB) using only $O(\log m)$ random bits. We shall need following lemma, which can be attributed to \[BE11\]. We repeat the proof of \[BE11\] for the sake of completeness.
Lemma 1.15. Let \( N = \lceil \frac{(\log n) n}{\epsilon} \rceil \) and \( p \) be a prime between \( N \) and \( 2N \). Let \( v_k \) denote the vector \((1, k \mod p, k^2 \mod p, \ldots, k^{n-1} \mod p)\), here \( k \in [p] \).

Also, let \( v_{kl} \) denote the \( l \)-th component of \( v_k \). Then for any non-zero \( m \)-sparse multivariate polynomial \( P(x_1, x_2, \ldots, x_n) \),

\[
\Pr_{k \in [p]} [P(x^{v_{k1}}, x^{v_{k2}}, \ldots, x^{v_{kn}}) \neq 0] \geq 1 - \epsilon
\]

Proof. \( P(x_1, x_2, \ldots, x_n) \) is of the following form.

\[
P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} c_l \prod_{j=1}^{n} x_j^{a_{ij}}
\]

Define \( u_{ij} = (\alpha_{1i} - \alpha_{1j1}, \alpha_{ij2} - \alpha_{ij2}, \ldots, \alpha_{ijn} - \alpha_{ijn}) \) for \( 1 \leq i < j \leq m \). \( u_{ij} \) is a non-zero vector in \( \mathbb{Z}^n \). Let \( e \) be the power of \( p \) in greatest common divisor of components of \( u_{ij} \). \( u'_{ij} \) is the vector obtained by dividing components of \( u_{ij} \) by \( p^e \). By this construction, we have made sure that \( u'_{ij} \mod p \neq 0 \). We observe that \( u'_{ij} \cdot v_k \) is zero for at most \( n-1 \) different values of \( k \). Let us assume that \( u'_{ij} \cdot v_k \) is zero for at least \( n \) different values of \( k \). Let these values be \( k_1, k_2, \ldots, k_n \). Consider the univariate polynomial \( f(x) = \sum_{l=1}^{m} (u'_{ij})_l \cdot x^{l-1} \), here \( (u'_{ij})_l \) is the \( l \)-th component of \( u'_{ij} \). \( f(x) \) is a non-zero polynomial in \( \mathbb{F}_p[x] \) because \( u'_{ij} \mod p \neq 0 \). Since \( u'_{ij} \cdot v_k \) is zero for \( 1 \leq i \leq n \), we see that \( k_1, k_2, \ldots, k_n \) are roots of \( f(x) \) in \( \mathbb{F}_p[x] \). But a non-zero polynomial of degree \( n-1 \) can have at most \( n-1 \) roots in \( \mathbb{F}_p[x] \), a contradiction. Hence \( u'_{ij} \cdot v_k \) is zero for at most \( n-1 \) different values of \( k \). Note that if \( u'_{ij} \cdot v_k \) is non-zero then \( u_{ij} \cdot v_k \) is also non-zero. Hence \( u_{ij} \cdot v_k \) is also zero for at most \( n-1 \) different values of \( k \). Hence number of different values of \( k \) such that \( u_{ij} \cdot v_k \) is zero for any \( 1 \leq i < j \leq m \), is at most \( \binom{m}{2} (n-1) \). Thus

\[
\Pr_{k \in [p]} [\exists 1 \leq i < j \leq m \text{ such that } u_{ij} \cdot v_k = 0] \leq \binom{m}{2} (n-1) \leq \frac{(m)(n-1)}{p} \leq \epsilon
\]

Suppose we choose a \( k \) such that \( \not\exists 1 \leq i < j \leq m \) having \( u_{ij} \cdot v_k = 0 \), then \( P(x^{v_{k1}}, x^{v_{k2}}, \ldots, x^{v_{kn}}) \neq 0 \) because every monomial in \( P(x^{v_{k1}}, x^{v_{k2}}, \ldots, x^{v_{kn}}) \) will have different exponent. Therefore

\[
\Pr_{k \in [p]} [P(x^{v_{k1}}, x^{v_{k2}}, \ldots, x^{v_{kn}}) \neq 0] \geq 1 - \epsilon.
\]

Now we present our randomized algorithm using only \( O(\log m + \log d) \) random bits where the degree of \( P(x_1, x_2, \ldots, x_n) \) is bounded by \( d \).
Algorithm 1.2 Algorithm for bounded degree

Input: A black-box for a \( m \)-sparse multivariate polynomial \( P(x_1, x_2, \ldots, x_n) \in \mathbb{Q}[x_1, x_2, \ldots, x_n] \) having degree at most \( d \).

Output: Determine whether \( P(x_1, x_2, \ldots, x_n) \) is zero or not. If \( P(x_1, x_2, \ldots, x_n) \) is zero then with probability 1 output that \( P(x_1, x_2, \ldots, x_n) \) is zero, otherwise with probability \( \geq \frac{1}{2} \) output \( P(x_1, x_2, \ldots, x_n) \) is non-zero.

1. Let \( N = \lceil \frac{\log n}{\epsilon} \rceil \).
2. Find a prime \( p \) between \( N \) and \( 2N \) with success probability at least \( 1 - \epsilon \).
3. Choose \( k \) uniformly at random from \([p]\).
4. Compute \( v_k = (1, k \mod p, k^2 \mod p, \ldots, k^{n-1} \mod p) \).
5. Let \( T = \lceil \frac{\log n}{\epsilon} \rceil \).
6. Find a prime \( q \) between \( T \) and \( 2T \) with success probability at least \( 1 - \epsilon \).
7. Choose \( t \) uniformly at random from \([q]\). Let \( r = t - 1 \).
8. Query black-box at the point \( P^r_r = (r^{v_{k1}} \mod q, r^{v_{k2}} \mod q, \ldots, r^{v_{kn}} \mod q) \). If answer is zero then output “\( P(x_1, x_2, \ldots, x_n) \) is zero” else output “\( P(x_1, x_2, \ldots, x_n) \) is non-zero”.

Theorem 1.16. Algorithm 1.2 solves PITQBB with success probability \( 1 - 4\epsilon \), in time \( \text{poly}(n, \log m, \log d, \log \frac{1}{\epsilon}) \) and uses \( O(\log m + \log d + \log \frac{1}{\epsilon}) \) random bits.

Proof. Let \( P'(x) = P(x^{v_{k1}}, x^{v_{k2}}, \ldots, x^{v_{kn}}) \). Using Lemma 1.15 \( P'(x) \in \mathbb{Q}[x] \) is a non-zero univariate polynomial with probability at least \( \geq 1 - \epsilon \). Since the degree of \( P(x_1, x_2, \ldots, x_n) \) is bounded by \( d \), degree of \( P'(x) \) is bounded by \( \text{pden} \). Let \( R(x) \in \mathbb{Z}[x] \) be the polynomial obtained by multiplying \( P'(x) \) with the least common multiple of denominators of all coefficients of \( P'(x) \). \( R(x) \) is non-zero iff \( P'(x) \) is non-zero. Also for any \( a \in \mathbb{Z} \), \( R(a) = 0 \) iff \( P'(a) = 0 \). Let \( \epsilon \) be the power of \( q \) in greatest common divisor of coefficients of \( R(x) \). Let \( R'(x) \) is the polynomial obtained by dividing coefficients of \( R(x) \) by \( q^\epsilon \). By this construction, we have made sure that \( R'(x) \mod q \neq 0 \). Since \( q \) is greater than degree of \( R'(x) \), we get that \( R'(x) \) is non-zero polynomial in \( \mathbb{F}_q[x] \) with probability at least \( \geq 1 - \epsilon \). Conditional on the fact that \( R'(x) \) in non-zero polynomial in \( \mathbb{F}_q[x] \), we know that it can have at most \( \text{pden} \) roots in \( \mathbb{F}_q \). Hence probability that \( r \) is a root of \( R'(x) \) is at most \( \frac{\text{pden}}{q} \leq \epsilon \). Using Lemma 1.12 we see that \( R'(r) \neq 0 \) with
probability at least $\geq 1 - 2\epsilon$. And if $R'(r) \neq 0$ in $F_q$ then it follows that $P(r^{v_k} \mod q, \ldots, r^{v_k} \mod q) \neq 0$. Hence

$$\Pr_{k \in [p], r \in [q]} [P(r^{v_k} \mod q, \ldots, r^{v_k} \mod q) \neq 0] \geq 1 - 2\epsilon.$$ 

Now we show the desired upper bound on random bits used by Algorithm 1.2. We use randomness at Steps 2, 3, 6, and 7. We want to succeed with probability at least $\geq \Pr$ in Step 2. Using Lemma 1.9 this can be done using $O(\log (\frac{n^4}{\epsilon}) + \log \frac{1}{\epsilon}) = O(\log m + \log \frac{1}{\epsilon})$ random bits. Here we are using the assumption that $n \leq m$. Step 2 uses $|\log p| = O(\log N) = O(\log m + \log \frac{1}{\epsilon})$ random bits. We also want to succeed with probability at least $\geq 1 - \epsilon$ in Step 6. Using Lemma 1.9 this can be done using $O(\log (\frac{\log m}{\epsilon}) + \log \frac{1}{\epsilon}) = O(\log m + \log d + \log \frac{1}{\epsilon})$ random bits. Step 7 uses $|\log q| = O(\log T) = O(\log m + \log d + \log \frac{1}{\epsilon})$ random bits. Hence the total number of random bits used are $O(\log m + \log d + \log \frac{1}{\epsilon})$.

Now we show the desired upper bound on time consumed by Algorithm 1.2. Using Lemma 1.9. Step 2 works in time $\text{poly}(\log N, \log \frac{1}{\epsilon}) = \text{poly}(\log m, \log \frac{1}{\epsilon})$. Using Lemma 1.8, each component of $v_k$ can be computed in time $O(\log n \cdot (\log p)^2) = \text{poly}(\log m, \log \frac{1}{\epsilon})$ time. Hence $v_k$ can be computed in time $n \cdot \text{poly}(\log m, \log \frac{1}{\epsilon}) = \text{poly}(n, \log m, \log \frac{1}{\epsilon})$ time. Hence Step 4 works in time $\text{poly}(n, \log m, \log \frac{1}{\epsilon})$. Using Lemma 1.9, Step 6 works in time $\text{poly}(\log T, \log \frac{1}{\epsilon}) = \text{poly}(\log m, \log d, \log \frac{1}{\epsilon})$. In Step 8, we compute $O(\log p \cdot (\log q)^2) = \text{poly}(\log m, \log d, \log \frac{1}{\epsilon})$ time to compute each component of the point $P^v$. Hence Step 8 works in time $n \cdot \text{poly}(\log m, \log d, \log \frac{1}{\epsilon})$ time.

Since each of the steps 2 and 6 fail with probability at most $\epsilon$, Algorithm 1.2 fails with probability at most $4\epsilon$. This is because we already showed that $Pr_{k \in [p], r \in [q]} [P(r^{v_k} \mod q, \ldots, r^{v_k} \mod q) \neq 0] \geq 1 - 2\epsilon$. 

**Corollary 1.17.** For polynomials having degree bounded by $\text{poly}(m)$, Algorithm 1.2 solves PITQBB in the time $\text{poly}(n, \log m, \log \frac{1}{\epsilon})$ with success probability $1 - 4\epsilon$ and uses $O(\log m + \log \frac{1}{\epsilon})$ random bits.

**Remark 1.18.** Klivans and Spielman [KS01] also devised a randomized algorithm for PITRBB using $O(\log m + \log d + \log \frac{1}{\epsilon})$ random bits and running in the time $\text{poly}(n, \log d, \log \frac{1}{\epsilon})$. Our algorithm for PITQBB achieves the same performance but uses a different approach.

### 1.4.3 Reducing Random bits

In this section, we show a randomized algorithm for PITRBB which uses $O(\frac{\log^2 m}{\log \log m})$ random bits, thus improving upon the $O(\log^2 m)$ random bits
upper bound shown by Bläser and Engels [BE11]. First, we need the following lemma, this is the sparse analogue of the famous Schwartz-Zippel lemma.

**Theorem 1.19 (BE11).** Let $P(x_1, x_2, \ldots, x_n)$ be an $m$-sparse non-zero multivariate polynomial. Let $S$ be a finite subset of positive numbers. Then

$$|\{s \in S^n \mid P(s) = 0\}| \leq mn \cdot |S|^{n-1}.$$

Hence

$$Pr_{s \in S^n}[P(s) = 0] \leq \frac{mn}{|S|}.$$

**Proof.** We prove it by induction on $n$. For $n = 1$, number of positive roots of any non-zero univariate $m$-sparse polynomial is bounded by $m - 1$ using Descartes’ rule of signs. Hence for $n = 1$, $|\{s \in S \mid P(s) = 0\}| \leq (m - 1)$.

Assume induction hypothesis for $n - 1$. Let $P(x_1, x_2, \ldots, x_n)$ be of the following form.

$$P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{t} x_1^{\alpha_i} \cdot P_i(x_2, x_3, \ldots, x_n)$$

Assume that $P_i(x_2, x_3, \ldots, x_n)$ has $k_i$ monomials, we have $\sum_{i=1}^{t} k_i = m$. Here $t \leq m$ and also $k_i \leq m$. Consider the set $X = \{s \in S^{n-1} \mid P_1(s) = 0\}$. By induction hypothesis, $|X| \leq k_1(m - 1)|S|^{n-2} \leq m(n - 1)|S|^{n-2}$. Hence number of non-zeros of $P_1(x_2, x_3, \ldots, x_n)$ in $S^{n-1}$ is at least $(|S|^{n-1} - m(n - 1)|S|^{n-2})$. For each of these non-zeros $(s_2, s_3, \ldots, s_n)$, there are at least $(|S| - m)$ values $s_1$ of $x_1$ in $S$ such that $(s_1, s_2, \ldots, s_n) \in S^n$ is a non-zero of $P(x_1, x_2, \ldots, x_n)$. This follows from Descartes’ rule of signs. Hence

$$|\{s \in S^n \mid P(s) \neq 0\}| \geq (|S|^{n-1} - m(n - 1)|S|^{n-2})(|S| - m) \geq |S|^{n-2}(|S|^2 - mn|S| + m^2(n - 1)) \geq |S|^{n-2}(|S|^2 - mn|S|) \geq |S|^{n-1}(|S| - mn)$$

Hence

$$|\{s \in S^n \mid P(s) = 0\}| \leq |S|^n - |S|^{n-1}(|S| - mn) = mn \cdot |S|^{n-1}.$$
CHAPTER 1. POLYNOMIAL IDENTITY TESTING

Theorem 1.19 immediately hints to a randomized algorithm for PITRBB. If we select \( n \) numbers \( a_1, a_2, \ldots, a_n \) uniformly at random from the set \( \left[ \frac{mn}{\epsilon} \right] \) then for any non-zero \( m \)-sparse polynomial \( P(x_1, x_2, \ldots, x_n) \), we have that probability of \( P(a_1, a_2, \ldots, a_n) \) being zero is at most \( \epsilon \). This method trivially works in time \( \text{poly}(n, \log m, \log \frac{1}{\epsilon}) \). But it consumes \( n \cdot \log \left[ \frac{mn}{\epsilon} \right] \) random bits. We want to get a better upper bound on random bits. Suppose we had \( t < n \) variables instead of \( n \), then we would need \( t \cdot \log \left[ \frac{mn}{\epsilon} \right] \) random bits.

Bläser and Engels [BE11] described a randomized transformation which reduces the number of variables to \( O(\log \frac{mn}{\epsilon}) \), thus obtaining an upper bound of \( O(\log^2 \frac{mn}{\epsilon}) \) random bits consumed. This transformation is a randomized transformation which preserves the non-zero-ness of the polynomial with a high probability. Also, it reduces number of variables to \( O(\log \frac{mn}{\epsilon}) \) while keeping number of monomials at most \( m \). We slightly modify this transformation to reduce number of variables to \( O(\log \frac{mn}{\epsilon}) \), while preserving other properties of this transformation. Lemma 1.20 formalizes this argument.

**Lemma 1.20.** Let \( P(x_1, x_2, \ldots, x_n) \) be a \( m \)-sparse non-zero multivariate polynomial. Let \( N = \left[ \frac{mn}{\epsilon} \right] \) and \( p \) (not a power of 2) be a prime between \( N \) and \( 2N \). Let \( v_i \) denote the vector \((1, i \mod p, i^2 \mod p, \ldots, i^{n-1} \mod p)\), here \( i \in [p] \). Let \( v_{ij} \) denote the \( j \)-th component of \( v_i \). Let \( v_{ijk} \) be the \( k \)-th digit of \( v_{ij} \), when \( v_{ij} \) is represented in base \( \lfloor \log m \rfloor \). Consider the polynomial \( P'(y_1, y_2, \ldots, y_t) = P(\prod_{l=1}^{t} y_l^{v_{11}}, \prod_{l=1}^{t} y_l^{v_{22}}, \ldots, \prod_{l=1}^{t} y_l^{v_{1n^t}}) \), here \( t = \lfloor \log \lfloor \log m \rfloor \rfloor p \). Then

\[
\Pr_{i \in [p]} [P'(y_1, y_2, \ldots, y_t) \neq 0] \geq 1 - \epsilon.
\]

**Proof.** Assume that \( P'(y_1, y_2, \ldots, y_t) \) is zero with probability \( > \epsilon \). Consider the polynomial \( R(x) = P'(x, x^{\lfloor \log m \rfloor}, x^{[(\log m)^2]}, \ldots, x^{[(\log m)^{t-1}]} \). With the assumption \( P'(y_1, y_2, \ldots, y_t) \) is zero with probability \( > \epsilon \), \( R(x) \) is also zero with probability \( > \epsilon \). But

\[
R(x) = P'(x, x^{\lfloor \log m \rfloor}, x^{[(\log m)^2]}, \ldots, x^{[(\log m)^{t-1}]} \\
= \prod_{l=1}^{t} x^{[(\log m)^{l-1}]} v_{11}^{v_{11}}, \prod_{l=1}^{t} x^{[(\log m)^{l-1}]} v_{22}^{v_{22}}, \ldots, \prod_{l=1}^{t} x^{[(\log m)^{l-1}]} v_{n^t}^{v_{n^t}} \\
= P(x^{\sum_{k=1}^{t} v_{1k}^{(\log m)^{k-1}}}, x^{\sum_{k=1}^{t} v_{2k}^{(\log m)^{k-1}}}, \ldots, x^{\sum_{k=1}^{t} v_{n^t k}^{(\log m)^{k-1}}}) \\
= P(x^{v_1}, x^{v_2}, \ldots, x^{v_m})
\]

Using Lemma 1.13, \( R(x) = P(x^{v_1}, x^{v_2}, \ldots, x^{v_m}) \) is zero with probability at most \( \epsilon \). Hence assumption that \( P'(y_1, y_2, \ldots, y_t) \) is zero with probability \( > \epsilon \), is wrong. Thus \( P'(y_1, y_2, \ldots, y_t) \) is a zero polynomial with probability at most \( \epsilon \). Thus
Determine whether with success probability $1 - \epsilon$.

Now we are ready to describe our algorithm which improves upon the upper bound on number of random bits presented in [BE11].

**Algorithm 1.3** Algorithm with better upper bound on random bits

**Input:** A black-box for an $m$-sparse multivariate polynomial $P(x_1, x_2, \ldots, x_n) \in \mathbb{R}[x_1, x_2, \ldots, x_n]$.

**Output:** Determine whether $P(x_1, x_2, \ldots, x_n)$ is zero or not. If $P(x_1, x_2, \ldots, x_n)$ is zero then with probability 1 output that $P(x_1, x_2, \ldots, x_n)$ is zero, otherwise with probability $\geq 1 - 3\epsilon$ output $P(x_1, x_2, \ldots, x_n)$ is non-zero.

1. Let $N = \lceil \frac{(m^2)n}{\epsilon} \rceil$.
2. Find a prime $p$ between $N$ and $2N$, $p$ is not a power of 2 here.
3. Choose $i$ uniformly at random from $[p]$ with success probability at least $1 - \epsilon$.
4. Compute $v_i = (1, i \mod p, i^2 \mod p, \ldots, i^{n-1} \mod p)$.
5. $t = \lceil \log \frac{\log m}{\log p} \rceil$. Let $v_{ijk}$ be the $k$-th digit of $v_{ij}$ when $v_{ij}$ is represented in base $\lceil \log m \rceil$, i.e., $v_{ij} = \sum_{k=1}^{t} v_{ijk} \cdot (\lceil \log m \rceil)^{k-1}$. Here $v_{ij} = i^{j-1} \mod p = j$-th component of $v_i$.
6. Choose $t$ numbers $a_1, a_2, \ldots, a_t$ uniformly at random from $[\lceil \frac{mt}{\epsilon} \rceil]$.
7. Query black-box at the point $P_a^\ast = (\prod_{l=1}^{t} a^{v_{il}}, \prod_{l=1}^{t} a^{v_{i2l}}, \ldots, \prod_{l=1}^{t} a^{v_{i2tl}})$. If answer is zero then output “$P(x_1, x_2, \ldots, x_n)$ is zero” else output “$P(x_1, x_2, \ldots, x_n)$ is non-zero”.

**Theorem 1.21.** Algorithm $\blacksquare$ solves PITRBB in time $\text{poly}(n, \log m, \log \frac{1}{\epsilon})$ with success probability $1 - 3\epsilon$ and uses $O(\frac{\log^2 m}{\log \log m})$ random bits.

**Proof.** First we show the correctness of Algorithm $\blacksquare$. Consider the polynomial $P'(y_1, y_2, \ldots, y_t) = P(\prod_{l=1}^{t} y_{1l}^{v_{1il}}, \prod_{l=1}^{t} y_{1l}^{v_{i2l}}, \ldots, \prod_{l=1}^{t} y_{1l}^{v_{i2tl}})$. Using Lemma $\blacksquare$ we observe that $P'(y_1, y_2, \ldots, y_t)$ is a zero polynomial with probability at most $\epsilon$. So we have made sure that $P'(y_1, y_2, \ldots, y_t)$ is a non-zero polynomial with high probability. Using Theorem $\blacksquare$ if we select $t$ numbers $a_1, a_2, \ldots, a_t$ uniformly at random from $[\frac{mt}{\epsilon}]$ then $P'(a_1, a_2, \ldots, a_t)$ is zero with probability at most $\epsilon$. Note that
Algorithm 1.3. Using Lemma 1.9, Step 2 works in time \( \text{poly} \). We can compute \( \prod_{l=1}^{t} a_l^{v_{il}} \) each such conversion can be done in time \( \text{poly} \). Hence Step 3 uses \( \lceil \log p \rceil = O(\log N) = \text{poly}(\log m + \log \frac{1}{\epsilon}) \) random bits. Step 6 uses \( O(\frac{\log \frac{m}{\epsilon}}{\log \log m} \cdot \text{poly}(\log m + \log \frac{1}{\epsilon})) \) random bits. This is equal to \( O(\frac{\log^2 \frac{m}{\epsilon}}{\log \log m}) \).

Thus, total number of random bits used are \( O(\frac{\log^2 \frac{m}{\epsilon}}{\log \log m}) \).

Lastly, we need to show the claimed upper bound on time consumed by Algorithm 1.3. Using Lemma 1.9, Step 2 works in time \( \text{poly}(\log N, \log \frac{1}{\epsilon}) = \text{poly}(\log m, \log \frac{1}{\epsilon}) \). Using Lemma 1.8, each component of \( v_l \) can be computed in time \( O(\log n \cdot (\log p)^2) = \text{poly}(\log m, \log \frac{1}{\epsilon}) \) time. Hence \( v_l \) can be computed in time \( n \cdot \text{poly}(\log m, \log \frac{1}{\epsilon}) = \text{poly}(n, \log m, \log \frac{1}{\epsilon}) \) time. Hence Step 4 works in time \( \text{poly}(n, \log m, \log \frac{1}{\epsilon}) \). Step 5 just converts \( v_{ij} \) to base \( \lfloor \log m \rfloor \), each such conversion can be done in time \( \text{poly}(\log m, \log \frac{1}{\epsilon}) \) time. Hence Step 5 works in \( n \cdot \text{poly}(\log m, \log \frac{1}{\epsilon}) = \text{poly}(n, \log m, \log \frac{1}{\epsilon}) \) time. Step 6 works in time \( O(t \cdot \log (\frac{\log m}{\epsilon})) \) time, which is \( \text{poly}(\log m, \log \frac{1}{\epsilon}) \). In Step 7 we need to compute point \( P^\star_a \) from \( a_i^l \)’s and \( v_l \). The \( j \)-th component of \( P^\star_a \) is \( \prod_{l=1}^{t} a_l^{v_{il}} \). Since \( v_{ij} < \lfloor \log m \rfloor \) and bit length of \( a_l \) is \( O(\log m + \log \frac{1}{\epsilon}) \), using Corollary 1.7 we can compute \( a_l^{v_{il}} \) in time \( O((\log m + \log \frac{1}{\epsilon})^2 \cdot \log^2 m) \). For every \( l \in [t] \), bit length of \( a_l^{v_{il}} \) is \( O(\log m \cdot (\log m + \log \frac{1}{\epsilon})) \). Using Corollary 1.6 we can compute \( \prod_{l=1}^{t} a_l^{v_{il}} \) in time \( O(t^2 \cdot \log^2 m \cdot (\log m + \log \frac{1}{\epsilon})^2) = \text{poly}(\log m, \log \frac{1}{\epsilon}) \). Hence \( P^\star_a \) can be computed in time \( n \cdot \text{poly}(\log m, \log \frac{1}{\epsilon}) = \text{poly}(n, \log m, \log \frac{1}{\epsilon}) \).
Chapter 2

Polynomial Interpolation

We present a deterministic algorithm to interpolate any $m$-sparse $n$-variate polynomial. This algorithm uses $\text{poly}(n, m, \log H, \log d)$ bit operations. Our algorithm works over the integers. Here $H$ is bound on the magnitude of the coefficient values of the given polynomial. This running time is polynomial in the output size. Our algorithm only requires black box access to the given polynomial, albeit in a special form. To our knowledge, this is the first algorithm in which the number of the bit operations have logarithmic dependence on the degree. As an easy consequence, we obtain an algorithm to interpolate polynomials represented by arithmetic circuits.

2.1 Introduction

Polynomial interpolation has always been an important problem in mathematics and computer science. Interpolation techniques by Lagrange and Newton have been very useful for interpolating univariate polynomials over fields of characteristic zero. There also has been lot of work on interpolating multivariate polynomials over fields of any characteristic. Zippel [Zip79] presented a randomized algorithm for polynomial interpolation. This inspired a lot of further research for finding deterministic algorithms for polynomial interpolation. An important technique for devising deterministic algorithms for polynomial interpolation was provided by Grigoriev and Karpinski [GK86], in their work on finding matchings for bipartite graphs having bounded permanent. Ben-Or and Tiwari [BO88] developed a deterministic algorithm for interpolating $m$-sparse multivariate polynomial in the black-box model using the ideas of Grigoriev and Karpinski [GK86]. This algorithm cleverly chooses the following points for evaluation

$$(p_1^i, p_2^i, \ldots, p_n^i)$$

where $i$ is in range from 0 to $2m - 1$ and $p_1$, $p_2$, $\ldots$, $p_n$ are first $n$ primes. The algorithm by Ben-Or and Tiwari uses $m^2(\log^2 m + \log nd)$
ring operations and $2m$ evaluations of the polynomial. But this operation count is in the algebraic RAM model and not in the traditional Turing model. Interpolation over finite fields have proved to be more difficult because $x^q \equiv x$ in $\mathbb{F}_q$. Polynomial interpolation has been extensively studied over finite fields in [Wer94, GKS90, CDGK91]. Grigoriev, Karpinski and Singer [GKS90] devised the first NC algorithm for interpolating $m$-sparse polynomials over finite fields. Their algorithm can be used to interpolate any $m$-sparse and $n$-variate polynomial in $O(\log^3(nm))$ Boolean parallel time, using $O(n^2m^6\log^2(nm))$ processors. Clausen, Dress, Grabmeier and Karpinski [CDGK91] showed that the number of queries to the black-box depend upon the degree of the extension field in which we make queries. For finite fields, if the degree of extension is equal to number of variables then we can interpolate any $m$-sparse polynomial with $m + 1$ queries. Klivans and Spielman [KS01] discovered an algorithm for polynomial interpolation over fields of large characteristic. The reader should note that sparsity of the polynomial is very important when considering the interpolation. If number of monomials $m$ is large, then the description of the polynomial is large, too. Representing an $m$-sparse $n$-variate polynomial as sum of monomials consumes $O(m \cdot (\log H + n \log d))$ space, $H$ being the bound on the magnitude of coefficient values and $d$ being the bound on the degree of every variable. So our algorithm running in the time $\text{poly}(n, m, \log H, \log d)$, is really optimal in the sense that the running time is polynomial in the output size.

### 2.2 New Black-box Model

Our algorithm uses access to the black-box in a new way. In the traditional black-box model we are given a black-box which represents a multivariate polynomial $P(x_1, x_2, \ldots, x_n) \in \mathbb{R}[x_1, x_2, \ldots, x_n]$, here $\mathbb{R}$ is the underlying ring. The black-box takes a point $(a_1, a_2, \ldots, a_n) \in \mathbb{R}^n$ as input and returns $P(a_1, a_2, \ldots, a_n) \in \mathbb{R}$ as output. Any interpolation algorithm in this model asks for value of $P$ at some set of points and after that it has to output $P$ as a list of coefficients along with corresponding monomials.

![Figure 2.2.1: Traditional Black-Box Model](image)

As mentioned earlier, in our case $\mathbb{R} = \mathbb{Z}$. In our new black-box model, there are two inputs instead of one. The first input is same as the traditional model, i.e., a point $(a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$. The second input is a positive number $N \in \mathbb{N}^+$. As an output, we get $P(a_1, a_2, \ldots, a_n) \mod N$ instead of $P(a_1, a_2, \ldots, a_n)$. 
CHAPTER 2. POLYNOMIAL INTERPOLATION

17

\[ P(x_1, x_2, \ldots, x_n) \]

\[ P(a_1, a_2, \ldots, a_n) \mod N \]

Figure 2.2.2: Our Black-Box Model

This new black-box model is a very natural extension of the traditional black-box model. The first reason for this is that we do not get any extra information in this new black-box model. The traditional black-box model can easily be simulated in this new black-box model, we just need to choose \( N \) (in Figure 2.2.2) large enough. Or one can also use Chinese remaindering to compute \( P(a_1, a_2, \ldots, a_n) \) by computing \( P(a_1, a_2, \ldots, a_n) \mod N \) for many relatively prime \( N \). It is true even the other way round. We can easily simulate this new black-box model by using the traditional black-box model. Secondly, if we want to have an interpolation algorithm which works in time sub-linear in the degree \( d \), then we can not use the traditional black-box model. This is because \( P(a_1, a_2, \ldots, a_n) \) will have bit size of \( \Omega(d) \) almost always. Lastly, this new black-box model is generalized version of arithmetic circuits. It is easy to simulate the new black-box model when the polynomial is given as an arithmetic circuit. We shall make this last point more precise in Section 2.7.

2.3 Our approach

It is not at all trivial to see why this new black-box model should help us to get faster algorithms for interpolation, since the traditional black-box model has all the information as the new black-box model. One thing that this new black-box model has in its favor is the fact that by choosing \( N \) (in Figure 2.2.2), we can control the size of output returned by the black-box. Still, it is not clear how to use this to devise faster interpolation algorithms. We precisely use this feature of the new black-box model in a novel way. Our approach has two main components. The first component finds some relatively small prime number \( q_0 \) such that projection of polynomial (which we are trying to interpolate) in \( \mathbb{F}_{q_0}[x] \) has the same number of monomials as the polynomial itself. For this, we can not just consider any prime. We shall use primes generated by arithmetic progressions for this purpose. The second component uses this prime \( q_0 \) to find sufficiently many primes \( p \) of a very special form. Primes \( p \) in this phase are also generated by arithmetic progressions. But we choose these arithmetic progressions in a subtle way so that computing the polynomial from its projections in \( \mathbb{F}_p[x] \) (for many primes \( p \)) is efficient. If we choose these primes \( p \) by some arbitrary strategy then it would not be clear how to compute the polynomial from its projections in \( \mathbb{F}_p[x] \). Our novel way of choosing these primes \( p \) in the second phase of
our algorithm, makes sure that it is easy to compute the polynomial from its projections.

2.4 Preliminaries

Here, some of the important results will also be duplicated from Chapter 1. We reproduce all the main results below to make this chapter self-contained. In this chapter, $\mathbb{Z}$ denotes the set of integers. $\mathbb{N}$ denotes the set of natural numbers. $\mathbb{N}^+$ denotes the set of positive integers. $\lfloor n \rfloor$ denotes the set $\{1, 2, \ldots, n\}$. We also use the notation $\tilde{O}(t)$ to denote the complexity $O(t \cdot (\log t)^{O(1)})$. $\mathbb{Z}_p$ denotes the quotient group $\mathbb{Z}/p\mathbb{Z}$. The degree of a polynomial is defined as the maximum degree of any variable. $F_p(x)$ denotes the polynomial $F(x) \mod p$. Note that $F_p(x)$ is the projection of $F(x)$ into $\mathbb{F}_p[x]$. More precisely, if $F(x) = \sum_{i=1}^{m} c_i x^{\alpha_i}$ then $F_p(x) = \sum_{i=1}^{m} (c_i \mod p) x^{\alpha_i \mod (p-1)}$.

Here we think of $c_i \mod p$ as an integer belonging to the set $\{0, 1, 2, \ldots, p-1\}$. Similarly for $\alpha_i \mod (p-1)$. More specifically, whenever the operator “mod $N$” appears in this chapter for some positive number $N$, we just think of the result of the operator as the remainder (a number in $\{0, 1, 2, \ldots, N-1\}$) when the operand is divided by $N$.

It is easy to see that if we interpolate $F(x)$ modulo a prime $p$ then we obtain $F_p(x)$ (due to Fermat’s little theorem). Whenever we say “time” in this paper, we mean the number of bit operations. Some of the results in this section can be found in [BHLD08].

**Fact 2.1.** Any integer $n \geq 1$ has at most $\log n$ distinct prime divisors.

**Fact 2.2.** The $k$-th prime number is of order $\Theta(k \log k)$.

Using the same notation as in [BHLD08], let $P(k)$ denote the smallest prime number in the arithmetic progression $\{jk + 1 \mid j \geq 1\}$. Linnik’s Theorem gives an unconditional upper bound on $P(k)$.

**Theorem 2.3** (Linnik’s Theorem [Lin44]). There is a constant $L > 1$ (called Linnik’s constant) such that $P(k) < k^L$ for every sufficiently large $k \geq k_0$.

The current best upper bound for $L$ is known to be 5.2 due to [Xyl09]. For a discussion on Linnik’s Theorem, reader is referred to [Rib96].

In the ensuing discussion, we would want that for different primes $q_1 \neq q_2$, we get that $P(q_1) \neq P(q_2)$. However this is not always true. But the following lemma makes sure that $P(q)$ cannot be the same for too many distinct primes $q$.

**Theorem 2.4** (Lemma 2 in [BHLD08]). Let $k_0$ be the constant mentioned in Linnik’s Theorem above. Let $q_1, q_2, \ldots, q_v$ be distinct primes greater than $k_0$ and $p$ be the prime such that $\forall i \in [v] : P(q_i) = p$. Then, $v \leq 5$. 

Proof. Since \( \forall i \in [v] P(q_i) = p \), we get that \( \forall i \in [v] q_i \mid (p-1) \). Hence \( q_1 q_2 \ldots q_v \mid (p-1) \). Therefore \( q_1 q_2 \ldots q_v < p \). It implies that \( q_1^v < q_1 q_2 \ldots q_v < p \). We also have that \( p < q_1^5 \). Therefore \( q_1^v < q_1^5 \). Since \( v \) is a positive integer, we get that \( v \leq 5 \). 

We would need the following result about modular computation. The reader is referred to [GG03] for details.

**Fact 2.5** (COROLLARY 4.7 in [GG03]). One arithmetic operation, that is, addition, multiplication, or division by an invertible element in \( \mathbb{Z}_p \) can be performed using \( O(\log^2 p) \) bit operations.

We shall be using the Chinese remainder theorem and gcd computations in the next section. We shall use the following Generalized Chinese remainder theorem and we shall also need an algorithm to find the solution in the case of generalized Chinese remainder theorem. We refer the reader to [BS96] for the following facts.

**Fact 2.6** (COROLLARY 4.2.4 in [BS96]). Let \( u, v \) be positive integers. We can compute the greatest common divisor of \( u \) and \( v \) using \( O(\log u \cdot \log v) \) bit operations.

**Lemma 2.7** (THEOREM 5.4.1 in [BS96]). For positive integers \( a, e \) and \( n \), there is an algorithm to compute \( a^e \mod n \) in \( O((\log e)(\log n)^2) \) bit operations, here \( 0 \leq a < n \).

**Theorem 2.8** (Generalized Chinese Remainder Theorem, THEOREM 5.5.5 in [BS96]). Let \( m_1, m_2, \ldots, m_k \) be positive integers. Then the system of congruences

\[
x \equiv x_i \mod m_i, 1 \leq i \leq k
\]

has a solution iff \( x_i \equiv x_j \pmod{\gcd(m_i, m_j)} \) for all \( i \neq j \). If the solution exists, it is unique \( \pmod{\text{lcm}(m_1, m_2, \ldots, m_k)} \).

**Lemma 2.9** (COROLLARY 5.5.6 in [BS96]). Let \( m_1, m_2, \ldots, m_k \) be positive integers, each \( \geq 2 \), define \( m = m_1 m_2 \ldots m_k \), and \( m' = \text{lcm}(m_1, m_2, \ldots, m_k) \). Given the system \( S \) of congruences

\[
x \equiv x_i \mod m_i, 1 \leq i \leq k
\]

we can determine if \( S \) has a solution, using \( O((\log m)^2) \) bit operations, and if so, we can find the unique solution \( \pmod{m'} \), using \( O((\log m)^2) \) bit operations.
2.5 Univariate interpolation

With all the machinery introduced in the last section, we are ready to describe our algorithm for interpolating univariate polynomials. For describing the algorithm, we need some definitions. In the following discussion, we always assume that we are trying to interpolate the following kind of univariate polynomial:

\[ F(x) = \sum_{i=1}^{m} c_i x^{\alpha_i}. \]

Here \(|c_i| \leq H\). And, \(d \geq \alpha_m \neq \alpha_{m-1} \neq \ldots \neq \alpha_1 \geq 0\). From now on, all the primes we encounter will be more than \(k_0\), the constant mentioned in Linnik’s Theorem.

2.5.1 Finding a Good prime

As we noted earlier, if we interpolate a polynomial \(F(x)\) modulo a prime \(p\) then we get all the coefficients modulo \(p\). This means that all the coefficients which are 0 modulo \(p\), vanish while interpolating modulo \(p\) and hence do not appear in \(F_p(x)\). Since we do not want to miss any coefficients for complete interpolation, we would want to avoid interpolating modulo any prime \(p\) such that some coefficient is 0 modulo \(p\). Also, we saw that we get all exponents modulo \((p-1)\). If the exponents of two different monomials are the same modulo \((p-1)\) then these monomials get merged into a single monomial while interpolating modulo \(p\). We would also like to avoid this situation. The following definitions formalize this notion.

Definition 2.10 (Coefficient bad prime). A prime \(q\) is called Coefficient bad for a polynomial \(F(x)\) if there exists some coefficient \(c_i\) such that \(c_i \equiv 0 \mod q\).

Lemma 2.11. There are at most \(m \log H\) Coefficient bad primes.

Proof. This lemma is a direct implication of the Fact \[2.1\] and the fact that \(|c_i| \leq H\). \hfill \Box

Definition 2.12 (Linnik Coefficient bad number (or prime)). A number (or prime) \(q\) is called Linnik Coefficient bad for a polynomial \(F(x)\) if \(P(q)\) is Coefficient bad prime for \(F(x)\).

Lemma 2.13. There are at most \(5m \log H\) Linnik Coefficient bad primes.

Proof. This lemma is a direct implication of the Theorem \[2.4\] and the Lemma \[2.11\]. \hfill \Box

Definition 2.14 (Power bad prime). A prime \(q\) is called Power bad for a polynomial \(F(x)\) if there exist \(1 \leq i \neq j \leq m\) such that \(\alpha_i \equiv \alpha_j \mod (q-1)\).
Definition 2.15 (LinnikPowerbad number (or prime)). A number (or prime) \( q \) is called LinnikPowerbad for a polynomial \( F(x) \) if \( P(q) \) is Powerbad prime for \( F(x) \).

Lemma 2.16. There are at most \( \binom{m}{2} \log d \) LinnikPowerbad primes.

Proof. By way of contradiction, assume that there are more than \( \binom{m}{2} \log d \) LinnikPowerbad primes. Let these primes be \( q_1, q_2, \ldots, q_k \), here \( k > \binom{m}{2} \log d \). Then we have that for all \( l \in [k] \) there are \( i \neq j \) such that \( \alpha_i \equiv \alpha_j \mod (P(q_l) - 1) \). Since \( k > \binom{m}{2} \log d \), by pigeonhole principle there exist \( t \) (more than \( \log d \)) primes \( r_1, r_2, \ldots, r_t \) in \( q_1, q_2, \ldots, q_k \) such that we have two different indices \( i \in [m] \) and \( j \in [m] \) satisfying \( \forall l \in [t] : \alpha_i \equiv \alpha_j \mod (P(r_l) - 1) \). Hence \( \forall l \in [t] : (P(r_l) - 1) | (\alpha_i - \alpha_j) \). This implies that \( \forall l \in [t] : r_l | (|\alpha_i - \alpha_j|) \). We have that \( |\alpha_i - \alpha_j| \leq d \). Using the Fact 2.1, \( |\alpha_i - \alpha_j| \) can have at most \( \log d \) distinct prime divisors. But we have \( t > \log d \) distinct prime divisors of \( |\alpha_i - \alpha_j| \). Hence our assumption that there are more than \( \binom{m}{2} \) LinnikPowerbad primes, is wrong. Thus there are at most \( \binom{m}{2} \log d \) LinnikPowerbad primes.

Definition 2.17 (Bad number (or prime)). A number (or prime) \( q \) is called Bad for a polynomial \( F(x) \) if it is LinnikCoefficientbad or LinnikPowerbad or both.

Lemma 2.18. There are at most \( \binom{m}{2} \log d + 5m \log H \) Bad primes.

Proof. Follows from the Lemma 2.13 and Lemma the 2.16

Definition 2.19 (Good number (or prime)). A number (or prime) \( q \) is called Good for a polynomial \( F(x) \) if it is not a Bad number (or prime).

Note that if we interpolate \( F(x) \) modulo a prime \( p \) then we obtain the polynomial \( F_p(x) \).

Suppose \( q \) is some Good prime. If we interpolate \( F(x) \) modulo \( P(q) \) then we shall get a list of all \( m \) coefficients modulo \( P(q) \) and corresponding powers modulo \( P(q) - 1 \). On the other hand, if \( q \) was a Bad prime then by definition of Bad primes, we would obtain less than \( m \) coefficients while interpolating modulo \( P(q) \). This happens because for a bad prime \( q \), some coefficient will be \( 0 \) modulo \( P(q) \) or two monomials will merge into one. This argument gives us a test to find Good primes. Since we know that there are at most \( \binom{m}{2} \log d + 5m \log H \) Bad primes, if we try more than \( \binom{m}{2} \log d + 5m \log H \) primes then we shall surely find a Good prime. And out of these primes, primes which give maximum number of coefficients are surely Good primes. But we need to make sure that interpolation alone does not take too much time. The following lemma makes sure that interpolation modulo a prime can be performed efficiently. From now on we use \( b = \binom{m}{2} \log d + 5m \log H \) to denote the maximum number of Bad primes.
**Theorem 2.20** (Theorem 5.1 in [GG03]). Let $\mathbb{F}$ be a field. Also, let $f(x) \in \mathbb{F}[x]$ be a polynomial of degree less than $n$. Given the value of $f(x)$ at $n$ distinct points $u_0, u_1, \ldots, u_{n-1} \in \mathbb{F}$, interpolation of $f(x)$ can be performed with $O(n^2)$ operations in $\mathbb{F}$.

**Proof.**

Algorithm 2.1 is $\tilde{O}(n^2)$ due to the Theorem 2.20. For this we need to know the value of $f(x)$ at $p$ distinct points in $\mathbb{F}$. The value of $F(x)$ modulo $p$ is equal to $F_p(x)$ at all points of $\mathbb{F}$. Since each operation in $\mathbb{F}$ can be performed in $O(\log^2 p)$ bit operations using Fact 2.5, interpolation $F_p(x)$ of $F(x)$ modulo a prime $p$ can be computed in time $O(p^2)$.

Now we describe the algorithm to find a *Good* prime.

**Algorithm 2.1** Algorithm to find a *Good* prime and exact value of $m$

**Input:** Black-box for polynomial $F(x)$.

**Output:** A *Good* prime and the exact value of number of monomials in $F(x)$.

1. Let $b = \binom{\binom{\binom{n}{2}}{2}}{2} \log d + 5m \log H$. Compute $b + 1$ primes $p_1 < p_2 < \ldots < p_{b+1}$ and also $P(p_1), P(p_2), \ldots, P(p_{b+1})$.

2. Interpolate $F(x)$ modulo $P(p_1), P(p_2), \ldots, P(p_{b+1})$ to obtain $F_{P(p_1)}(x), F_{P(p_2)}(x), \ldots, F_{P(p_{b+1})}(x)$.

3. Find any prime $p_i$ such that $F_{P(p_i)}(x)$ has maximum number of monomials among $F_{P(p_1)}(x), F_{P(p_2)}(x), \ldots, F_{P(p_{b+1})}(x)$.

4. Output $p_i$ as a *Good* prime and the number of monomials in $F_{P(p_i)}(x)$ as $m$.

**Claim 2.22.** Algorithm 2.1 finds a *Good* prime and $m$ in time $\tilde{O}(b^{2L+1})$.

**Proof.** Correctness follows from the earlier discussion. Since $b = \binom{\binom{\binom{n}{2}}{2}}{2} \log d + 5m \log H$, Step 1 find all the required primes in time $\tilde{O}(b^{L+1})$. Here we use the Fact 2.2 to make sure that $p_{b+1}$ (and hence $P(p_{b+1})$, due to the Theorem 2.3) is of absolute value $\tilde{O}(b^L)$. To check for primality of a number, we can use AKS algorithm [AKS04]. The total time of Step 2 is still $\tilde{O}(b^{L+1})$. Since all primes in Step 1 are of magnitude $\tilde{O}(b^{L+1})$, computing $F_{P(p_i)}(x)$ takes $\tilde{O}(b^{2L})$ due to Lemma 2.21. Hence Step 2 takes $\tilde{O}(b+1)\tilde{O}(b^{2L}) = \tilde{O}(b^{2L+1})$ time. Step 3 and 4 trivially take $\tilde{O}(b)$ time. Hence the total time taken by Algorithm 2.1 is $\tilde{O}(b^{2L+1})$. \qed
CHAPTER 2. POLYNOMIAL INTERPOLATION

From now on, let \( q_0 \) be the Good prime found by Algorithm 2.1 and also let \( g = P(q_0) - 1 \). Note that absolute value of \( g \) is \( \text{poly}(m, \log d, \log H) \). Now we shall not find Good primes but we shall try to find Good numbers of the form \( gp \), where \( p \) is some prime.

2.5.2 Finding many Good primes

**Definition 2.23** (Badg prime). A prime \( q \) is called Badg for polynomial \( F(x) \) if \( gq \) is Bad number. (Note that the term “Badg” depends on the number \( g \), which will be chosen as above and will be fixed throughout the remainder of this section.)

**Definition 2.24** (Goodg prime). A prime \( q \) is called Goodg for the polynomial \( F(x) \) if it is not Badg.

**Lemma 2.25.** There are at most \( b = \binom{m}{2} \log d + 5m \log H \) Badg primes.

**Proof.** It is easy to extend the proof of Lemma 2.13 to show that there are at most \( 5m \log H \) primes \( p \) such that \( gp \) is \( \text{LinnikCoefficientbad} \) number. And similarly proof of Lemma 2.16 can be extended to show that \( gp \) is \( \text{LinnikPowerbad} \) number for at most \( \binom{m}{2} \log d \) many primes \( p \). Thus there are at most \( \binom{m}{2} \log d + 5m \log H \) Badg primes.

From now on, we use \( t \) to denote the number \( \max\{\lceil \log H \rceil + 1, \lceil \log d \rceil \} \). The below written algorithm finds \( t \) Goodg primes \( q_1, q_2, \ldots, q_t \) in time \( \text{poly}(m, \log d, \log H) \) and also performs the interpolation of \( F(x) \) modulo \( P(gq_1), P(gq_2), \ldots, P(gq_t) \), i.e., it finds \( F_{P(gq_1)}(x), F_{P(gq_2)}(x), \ldots, F_{P(gq_t)}(x) \).

**Algorithm 2.2** Algorithm to find a \( t \) Goodg primes and corresponding interpolation

**Input:** Black-box for polynomial \( F(x) \), \( g = p(q_0) - 1 \) and \( m \). Here \( q_0 \) is the Good prime obtained by Algorithm 2.1 and \( m \) is the exact numbers of monomials in \( F(x) \) given by Algorithm 2.1. Let \( b = \binom{m}{2} \log d + 5m \log H \) and \( t = \max\{\lceil \log H \rceil + 1, \lceil \log d \rceil \} \).

**Output:** \( t \) Goodg primes \( q_1, q_2, \ldots, q_t \) and \( F_{P(gq_1)}(x), F_{P(gq_2)}(x), \ldots, F_{P(gq_t)}(x) \).

1. Compute \( b + t \) primes \( p_1 < p_2 < \ldots < p_{b+t} \) and also \( P(gp_1), P(gp_2), \ldots, P(gp_{b+t}) \).
2. Interpolate \( F(x) \) modulo \( P(gp_1), P(gp_2), \ldots, P(gp_{b+t}) \) to obtain \( F_{P(gp_1)}(x), F_{P(gp_2)}(x), \ldots, F_{P(gp_{b+t})}(x) \).
3. Find any \( t \) Goodg primes \( q_1, q_2, \ldots, q_t \) from \( p_1 < p_2 < \ldots < p_{b+t} \) such that \( F_{P(gq_1)}(x), F_{P(gq_2)}(x), \ldots, F_{P(gq_t)}(x) \) all have exactly \( m \) monomials.
Claim 2.26. Algorithm 2.2 finds $t$ Goodg primes and performs the corresponding interpolation in time $\tilde{O}(b^{2L^2+2L+1})$.

Proof. Correctness follows from Lemma 2.25. We just need to show the desired time bound. Step 1 takes $\tilde{O}(b^{2L^2+2L+1})$ time since $b + t$ and $g$ are of absolute value $O(b)$ and $\tilde{O}(b^L)$ respectively. In Step 2, since $P(gp_i)$ is also of absolute value $\tilde{O}(b^{2L^2+L})$, we can compute $F_P(gp_1)(x), F_P(gp_2)(x), \ldots, F_P(gp_{t+1})(x)$ in time $\tilde{O}(b^{2L^2+2L+1})$ due to Lemma 2.21. Therefore Step 2 takes $\tilde{O}(b^{2L^2+2L+1})$ time. Step 3 clearly takes $O(b)$ time. Hence total time taken by Algorithm 2.2 is $\tilde{O}(b^{2L^2+2L+1})$. \hfill \square

2.5.3 Main Algorithm for Interpolation

After applying Algorithm 2.2 we have found $F_P(gp_1)(x), F_P(gp_2)(x), \ldots, F_P(gp_{t+1})(x)$ for $t$ Goodg primes $q_1, q_2, \ldots, q_t$. We need to use $F_P(gp_1)(x), F_P(gp_2)(x), \ldots, F_P(gp_{t+1})(x)$ to compute $F(x)$.

Note that each $F_P(gp_i)(x)$ is nothing but a list of size $m$, each member of the list is a pair of some coefficient $\mod P(gp_i)$ and a corresponding power $\mod (P(gp_i) - 1)$. From now on, assume that each $F_P(gp_i)(x)$ is of the following form.

$$F_P(gp_i)(x) = \sum_{j=1}^{m} c_{ij} x^{\alpha_{ij}}$$

We can construct the $c_i$’s and the $\alpha_i$’s from the $c_{ij}$’s and the $\alpha_{ij}$’s using Chinese remaindering but we do not know which are the correct indexes to join. For example, we know that $c_1$ appears in $F_P(gp_1)(x)$ as some $c_{1j} = c_1 \mod P(gp_1)$ but we do not know the index $j$. Similarly for the powers $\alpha_i$’s. We know $\alpha_1$ appears in $F_P(gp_1)(x)$ as some $\alpha_{1j} = \alpha_1 \mod (P(gp_1) - 1)$ but we do not know the index $j$. But we shall show how to use the Chinese remaindering to compute $c_1$’s and $\alpha_i$’s from $c_{ij}$’s and $\alpha_{ij}$’s. We shall need the following lemma for this purpose.

Lemma 2.27. Let $u, v$ be distinct positive integers $\leq t$, and $s = \gcd(P(gp_u) - 1, P(gp_v) - 1)$. Then for any $j \in [m]$, there exists a unique $j' \in [m]$ such that $\alpha_{uj} \equiv \alpha_{uj'} \mod s$.

Proof. First we show the existence of $j'$. We know that $\alpha_{uj} \equiv \alpha_i \mod (P(gp_u) - 1)$ for some $i \in [m]$. Let $j'$ be such that $\alpha_{uj'} \equiv \alpha_i \mod (P(gp_v) - 1)$. Since $\alpha_i$ is a solution to

$$x \equiv \alpha_{uj} \mod (P(gp_u) - 1)$$
$$x \equiv \alpha_{uj'} \mod (P(gp_v) - 1),$$

by using Theorem 2.8 we need to have $\alpha_{uj} \equiv \alpha_{uj'} \mod \gcd(P(gp_u) - 1, P(gp_v) - 1))$. Therefore $\alpha_{uj} \equiv \alpha_{uj'} \mod s$. Now for uniqueness, by way
of contradiction assume that there exists $j'' \neq j'$ such that $\alpha_{v_j} \equiv \alpha_{v_{j''}} \mod s$. Then we get that $\alpha_{v_{j'}} \equiv \alpha_{v_{j''}} \mod s$. Let $k$ be such that $\alpha_{v_{j''}} \equiv \alpha_k \pmod{P(q_{g_0}) - 1}$. Thus $\alpha_{v_{j''}} \equiv \alpha_k \pmod{s}$. We also have $\alpha_{v_{j'}} \equiv \alpha_i \pmod{s}$. Altogether, we have the following congruences

$$\begin{align*}
\alpha_{v_{j'}} &\equiv \alpha_{v_{j''}} \pmod{s} \\
\alpha_{v_{j'}} &\equiv \alpha_i \pmod{s} \\
\alpha_{v_{j''}} &\equiv \alpha_k \pmod{s}.
\end{align*}$$

Hence $\alpha_i \equiv \alpha_k \pmod{s}$. Note that $g \mid s$. It implies that $\alpha_i \equiv \alpha_k \pmod{g}$. But this cannot happen because $g = (P(q_0) - 1)$ and $q_0$ was chosen to be a Good prime. Therefore there does not exist such $j'' \neq j'$. \hfill \Box

Lemma 2.27 gives us a method to find the correct $\alpha_{ij}$'s, from which we can recover all $\alpha_u$'s. Below described algorithm completes the interpolation algorithm. It takes $F_{P(q_{g_1})}(x), F_{P(q_{g_2})}(x), \ldots, F_{P(q_{g_0})}(x)$ and $P(q_{g_1}), P(q_{g_2}), \ldots, P(q_{g_0})$ as input and outputs $F(x)$. We shall also show that it runs in the time $\text{poly}(m, \log d, \log H)$. 
Algorithm 2.3 Algorithm to find $F(x)$

**Input:**
- $t$ Goodf, primes $q_1, q_2, \ldots, q_t$ and $F_{P(gq_1)}(x), F_{P(gq_2)}(x), \ldots, F_{P(gq_t)}(x)$.

**Output:** $F(x)$.

1. for $j = 1$ to $m$
   
   (a) for $i = 2$ to $t$
      
      i. $s_i = \gcd(P(gq_1) - 1, P(gq_i) - 1)$.
      
      ii. Find $k_{ij} \in [m]$ such that $\alpha_{ik_{ij}} \equiv \alpha_{1j} \mod s_i$.
   
   (b) Solve the following equations by Chinese remaindering to obtain $c_j$
      
      \[
      x \equiv c_{1j} \mod P(gq_1) \\
      x \equiv c_{2k_{2j}} \mod P(gq_2) \\
      \vdots \quad \vdots \\
      x \equiv c_{tk_{ij}} \mod P(gq_t)
      \]
   
   (c) Solve the following equations by Chinese remaindering to obtain $\alpha_j$
      
      \[
      x \equiv \alpha_{1j} \mod (P(gq_1) - 1) \\
      x \equiv \alpha_{2k_{2j}} \mod (P(gq_2) - 1) \\
      \vdots \quad \vdots \\
      x \equiv \alpha_{tk_{ij}} \mod (P(gq_t) - 1)
      \]
   
   (d) If $c_j > H$ then $c_j = c_j - \prod_{i=1}^{t} P(gq_i)$.

2. end for loop

3. Output $F(x)$ as $F(x) = \sum_{i=1}^{m} c_i x^{\alpha_i}$.

---

**Claim 2.28.** Algorithm 2.3 finds $F(x)$ in time $\tilde{O}(b^2)$.

**Proof.** Algorithm 2.3 applies Lemma 2.27 repeatedly. Step 1a finds the corresponding alignment of monomials. Here, $u$ of Lemma 2.27 is fixed to be 1 whereas $v$ runs from 2 to $t$. Step 1b computes the coefficient of the monomial whose alignment we found in Step 1a. Step 1c does the similar computation for the power of the corresponding monomial. Step 1d takes care of the fact
CHAPTER 2. POLYNOMIAL INTERPOLATION

that $c_i$ can be negative also. And since $t \geq \max\{\lceil \log H \rceil + 1, \lceil \log d \rceil \}$, we have $\prod_{i=1}^t P(gq_i) > 2H$ and $\text{lcm}\{P(gq_1) - 1, P(gq_2) - 1, \ldots, P(gq_t) - 1\} > d$. Hence correctness follows. We just have to show a running time upper bound of $\text{poly}(m, \log d, \log H)$. The main loop runs $m$ times. So we shall just show that one iteration of the loop takes $\text{poly}(m, \log d, \log H)$ time. Step 1a runs a loop $t$ times. Time taken by Step 1a is also $\tilde{O}(\log b)$ due to Fact 2.6 and the fact that $P(gq_i)$ is of size $\tilde{O}(b L^2 + L)$. Step 1a clearly works in time $\tilde{O}(\log b)$ as we just need to iterate over $m$ monomials of $F_{P(gq_i)}(x)$ and find the correct $k_{ij}$. Step 1b works in time $\tilde{O}(t^2 \cdot \log^2 b)$ because of the fact that $\log^2(\prod_{i=1}^t P(gq_i)) = O(t^2 \cdot \log^2 b) = \text{and Lemma 2.9}$. Similarly, Step 1c works in time $\tilde{O}(t^2 \cdot \log^2 b)$. The rest of the steps trivially work in time $O(b^2)$. Hence Algorithm 2.3 works in time $\tilde{O}(b^2)$.

Remark 2.29. Combining Algorithm 2.1, 2.2 and 2.3, we can see that interpolation of $F(x)$ can be performed in the time $\tilde{O}(b^{2L^2 + 2L + 1})$. Here $b = \binom{m}{2} \log d + 5m \log H$ and $L$ is the constant we encountered in Theorem 2.3.

2.6 Multivariate Interpolation

In the last section, we presented the algorithm to interpolate univariate polynomials which worked in time $\tilde{O}(b^{2L^2 + 2L + 1}) = \text{poly}(m, \log d, \log H)$. Now we show how to use that algorithm to interpolate multivariate polynomials. We need the following lemma for that, which goes back to Kronecker.

Lemma 2.30 (See e.g. Lemma 1 in [BHLD08]). Let $K$ be a ring. Let $f \in K[x_1, \ldots, x_n]$ be a polynomial of the degree at most $d$. Then the substitution $x_i \mapsto X^{(d+1)^i - 1}$ maps $f$ to a univariate polynomial $g \in K[X]$ of degree at most $(d + 1)^n - 1$ such that any two distinct monomials in $f$ map to distinct monomials in $g$. In particular, if $f$ is not identically zero in $K[x_1, \ldots, x_n]$, then $g$ is not identically zero in $K[X]$.

Substitution in Lemma 2.30 converts $f$ into a univariate of the degree at most $(d + 1)^n - 1$. And given $f$ in univariate, getting back multivariate $f$ is also easy. This gives us following algorithm to interpolate multivariate polynomials.
CHAPTER 2. POLYNOMIAL INTERPOLATION

Algorithm 2.4 Algorithm to interpolate Multivariate polynomial

**Input:** Black-box for polynomial \( P(x_1, x_2, \ldots, x_n) \).

**Output:** Polynomial \( P(x_1, x_2, \ldots, x_n) \).

1. Given the \( n \)-variate polynomial \( P(x_1, x_2, \ldots, x_i, \ldots, x_n) \) of degree at most \( d \), interpolate the univariate polynomial \( P'(X) = \sum_{i=0}^{d+1} \sum_{j=0}^{n-1} a_{ij} X^{(d+1)i-j} \) of degree at most \( (d+1)^n - 1 \).

2. Convert \( P' \) to \( P \) by computing the corresponding unique \( n \)-variate monomial for each univariate monomial in \( P \).

**Claim 2.31.** Algorithm 2.4 interpolates any \( n \)-variate polynomial of degree at most \( d \) and works in time \( \text{poly}(m,n,\log d, \log H) \).

**Proof.** Correctness follows from Lemma 2.30. The running time for interpolating \( P' \) is \( \text{poly}(m, \log((d+1)^n - 1), \log H) = \text{poly}(m,n,\log d, \log H) \) due to Lemma 2.28. We just need to make sure that the substitution \( x_i \mapsto X^{(d+1)i-1} \) can be computed efficiently. Note that all the numbers encountered in the algorithm for univariate polynomial interpolation are of absolute value \( \text{poly}(m,n,\log d, \log H) \). And for the purpose of asking the black-box, we just need to compute \( X^{(d+1)i-1} \mod N \) for some number \( N \) of absolute value \( \text{poly}(m,n,\log d, \log H) \). Hence \( X^{(d+1)i-1} \mod N \) can be computed in time \( \text{poly}(m,n,\log d, \log H) \) using Lemma 2.7. Step 2 is just a trivial conversion to base \((d+1)\). \( \square \)

2.7 Conclusion and applications

The reader should note that all the numbers used in our algorithm have magnitude \( \text{poly}(m,n,\log d, \log H) \). This makes our algorithm perfectly suited for interpolating sparse polynomials represented by arithmetic circuits. For an extensive discussion on arithmetic circuits, we refer reader to a recent survey on arithmetic circuits by Shpilka and Yehudayoff [SY10]. Let \( C \) be an arithmetic circuit of size \( s \) and description length \( l \) which computes the polynomial \( P(x_1, x_2, \ldots, x_n) \). Here the size \( s \) is the number of gates in the circuit and the description length \( l \) is the length of a reasonable encoding of the circuit as a binary string. In particular, \( 2^s \) is a bound on the degree of \( P \) and every constant appearing in the circuit is bounded by \( l \).

In our black-box model, we assumed that we can obtain the value of \( P \) at any point modulo some integer in unit time. But it is also easy to see that the value of \( P \) at points we encountered can be obtained in time \( \text{poly}(m,n,s,l, \log d, \log H) \). The following algorithm exactly performs this task.
CHAPTER 2. POLYNOMIAL INTERPOLATION

Algorithm 2.5 Algorithm to Evaluate circuit

**Input:** A description of length $l$ of a division free circuit $C$ (over $\mathbb{Z}$) of size $s$, $(a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ and $N \in \mathbb{N}^\ast$.

**Output:** $C(a_1, a_2, \ldots, a_n) \mod N$.

1. Evaluate every gate of $C$ modulo $N$.
2. Return result of output gate.

**Claim 2.32.** Algorithm 2.5 runs in time $\text{poly}(s, l, \log N, \max_i \log(|a_i| + 1))$.

**Proof.** Using Fact 2.31 evaluation at each gate of $C$ can be performed in time $O(s \cdot \log^2 N)$. We may need to perform more bit operations when we have to deal with constants in $C$ and $a_i$’s. Let $c$ be the constant used in $C$ of maximum magnitude. We have that $\log(|c| + 1) \leq l$. While evaluating gates which have a constant $c$ as input, we perform $O(s \cdot (\log(|c| + 1) \cdot \log N)^2)$ bit operations. Similarly while evaluating input gates where we have to deal with $a_i$’s, we perform $O(s \cdot (\log(|a_i| + 1) \cdot \log N)^2)$ bit operations. Since we have to do at most $s$ evaluations, Algorithm 2.5 runs in time $O(s^2 \cdot (\max_i \log(|a_i| + 1) \cdot \log N)^2)$ time. Since $\log c \leq l$, the whole algorithm takes $O(s^2 \cdot (\max_i \log(|a_i| + 1) \cdot l \cdot \log N)^2)$ time. Hence Algorithm 2.5 runs in time $\text{poly}(s, l, \log N, \max_i \log(|a_i| + 1))$. \hfill \square

Using Claim 2.32 it is easy to see that our algorithm for black-box interpolation can be used to interpolate sparse polynomial represented by circuits.

**Corollary 2.33.** A polynomial $P$ with at most $m$ monomials represented by an arithmetic circuit $C$ of size $s$ and description length $l$ can be interpolated in time $\text{poly}(m, \log H, l)$, here $H$ is bound on magnitude of coefficients of $P$.

**Proof.** Combining Algorithm 2.5 with Algorithm 2.4 gives us a method to interpolate $P$ in time $\text{poly}(s, l, m, n, \log H, \log d)$. Since we have that $d \leq 2^s$ and $n \leq s$, we get that whole algorithm runs in the time $\text{poly}(m, \log H, s, l)$. Also, we have $s \leq l$. Hence whole algorithm runs in the time $\text{poly}(m, \log H, l)$. \hfill \square

**Remark 2.34.** If we use faster algorithms for computing $F_p(x)$, such as $O(p)$ algorithm in [BCS97], then we can reduce the running time of our algorithm to $O(b^{L^2 + L + 1})$.

**Remark 2.35.** Recently, the constant $L$ of Theorem 2.3 has been improved to 5 in [Xyl11].
Bibliography


