Recovery of Exponents of Polynomials of High Degree

by

Jesse Elliott

Undergraduate Honours Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Science

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Fall 2016

Approval

Name:	**Jesse Elliott**
Degree:	**Bachelor of Science (Mathematics)**
Title:	***Recovery of Exponents of Polynomials of High Degree***
Examining Committee:	

Michael Monagan
Senior Supervisor
Professor _____

Marni Mishna
Professor _____

Abstract

We present a new algorithm for sparse multivariate polynomial interpolation. The algorithm is a modification of the Ben-Or and Tiwari sparse interpolation algorithm for efficient exponent recovery. While Kaltofen in [1] has shown that the Ben-Or/Tiwari method can be modified with a Kronecker substitution to reduce multivariate interpolation to a univariate interpolation, his method requires a prime $p > (d+1)^n$ where d is the maximum partial degree and n is the number of variables. For cases involving polynomials with many variables or high degree, p exceeds the machines register size and arithmetic becomes expensive. We present a modification that runs the algorithm modulo several machine primes. A modular extension for exponent recovery is challenging because the modular exponent images have arbitrary order. Our method involves choosing moduli that share a common divisor and sorting the images modulo this divisor.

Keywords: Sparse polynomial interpolation; Chinese remainder theorem; Ben-Or and Tiwari

Acknowledgements

I would like to thank my supervisor Michael Monagan for his encouragement, patience, and guidance. I would also like to thank Marni Mishna for the guidance that she provided. I thank my parents for their loyal support.

Table of Contents

Chapter 1

Introduction

This thesis presents a new modular algorithm for sparse multivariate polynomial interpolation. The algorithm is a modular extension of the Ben-Or and Tiwari sparse interpolation algorithm to efficiently recover large exponents.

1.1 Polynomial Interpolation

Polynomial interpolation is a fundamental part of many algorithms in computer algebra. These algorithms interpolate the polynomial from its values. Calculations with polynomials often involve *intermediate expressions swell*, which means that intermediate expressions require more space than the final result. Intermediate expressions can have exponential growth resulting in inefficient space complexities. Polynomial interpolation is used in computer algebra algorithms to avoid intermediate expression swell. Algorithms that involve operations with polynomials, such as computing the greatest common divisor of two polynomials or computing the determinant of a matrix of polynomials, ofter use polynomial interpolation.

In practice, polynomials in many variables are often *sparse*.

Definition 1.1.1. Let $f \in R[x_1, x_2, ..., x_n]$, where R is a ring and deg $f = d$. Let t denote the number of non-zero terms of f and $T_{\max}$ denote the maximum possible terms of f. The polynomial f is **sparse** if $t \ll T_{\max} = \binom{n+d}{d}$. If a polynomial is not sparse, we say it is **dense**.

The notation used in this definition was taken from [2].

Example 1. Let f, with deg $f = d$, be the following polynomial:

$$x_1^{d-1}x_2 + x_2^{d-1}x_3 + \ldots + x_n^{d-1}x_1 + 2.$$

The polynomial f has $n + 1$ terms. Since $n + 1 \ll \binom{n+d}{d}$, f is sparse.

Polynomials arising in practice generally are not as sparse as the example just given. A more typical example would be $\det V_n$, the determinant of the $n \times n$ Vandermonde matrix.

Example 2. Let

$$V_{10} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{10} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^9 & x_2^9 & \dots & x_{10}^9 \end{bmatrix}$$

be the Vandermonde matrix V_{10}. The determinant of V_{10} is a polynomial with 10! terms and degree 45 (i.e. $t = 10!$ and $d = 45$). The maximum number of terms of $\det V_{10}$ is $\binom{10+45}{45} = 29248649430$. The density ratio $t/T_{\max} = 0.000124$, and therefore $\det V_{10}$ is sparse.

Let p be a prime and let $f \in \mathbb{Z}[x_1, x_2, \dots, x_n]$ be a multivariate polynomial with $t > 0$ non zero terms, and let $d_i = \deg_{x_i} f$ be the partial degrees of f. We assume a *black box* representation of f. In the black box model we can only evaluate the polynomial at a point modulo a prime p, as shown in Figure 1.

Figure 1.

$$(a_1, a_2, ..., a_n) \in \mathbb{Z}_p^n \xrightarrow{\quad} \boxed{f} \xrightarrow{\quad} f(a_1, a_2, ..., a_n) \in \mathbb{Z}_p$$

On input $(a_1, a_2, ..., a_n) \in \mathbb{Z}_p^n$, the black box evaluates and outputs $f(a_1, a_2, ..., a_n) \in \mathbb{Z}_p$. As an example, the black box f could represent a matrix A of polynomials in $\mathbb{Z}[x_1, x_2, \dots, x_n]$, where the matrix entries may be explicit polynomials or black boxes of polynomials. In the black box model we usually assume we have degree bounds $\deg_{x_i} f$ and a term bound t. Methods for determining both exist. In this thesis both requirements will be relaxed.

1.2 Outline

In chapter 2 we present the new algorithm. The chapter begins with the development and context of the algorithm. Following this, we present an example of the algorithm. Next, the method for exponent recovery is presented. Finally, we present pseudo code and complexities. In chapter 3 we discuss smooth prime numbers and their importance and role in the algorithm. In chapter 4 we present conclusions.

Chapter 2

The Algorithm

2.1 Development and Context

2.1.1 Zippel's Sparse Interpolation Algorithm

Richard Zippel in 1979 developed the first sparse interpolation algorithm for multivariate polynomials. See [3]. The algorithm is probabilistic and uses the fact that if a polynomial evaluates to zero at a random point then there is a high probability that the polynomial is identically zero. In 1990 Zippel improved his algorithm by using special evaluation points in $\mathbb{Z}_p$. See [4]. The new algorithm is currently used in Maple, Magma, and Mathematica. For sparse polynomials, the worst case complexity requires $O(ndt)$ evaluations. For dense polynomials, the average case complexity requires $O(dt)$ evaluations.

2.1.2 Ben-Or and Tiwari's Sparse Interpolation Algorithm

The Ben-Or and Tiwari algorithm was the first deterministic polynomial time algorithm for sparse multivariate polynomial interpolation. The algorithm involves two main independent stages. Monomials are determined in the first stage and coefficients are determined in the second. The algorithm applies a technique for decoding BCH codes. The evaluations of the target polynomial are used to compute an error locating polynomial. This polynomial is referred to as the λ polynomial. In the first stage of the algorithm, the black box polynomial is evaluated at $(2^i, 3^i, \ldots, p_n^i)$ and the evaluations are used to determine the λ polynomial. The roots of the λ polynomial are used to determine the monomials. In the second stage of the algorithm, the coefficients are determined by solving a transposed Vandermonde system. See [5] for details.

Let

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, x_2 \ldots, x_n]$$

be the target polynomial, with $M_i = x_1^{d_{i1}} \cdot x_2^{d_{i2}} \cdot \ldots \cdot x_n^{d_{in}}$ and $a_i \in \mathbb{Z}\backslash\{0\}$. Let $d_i = \deg_{x_i} f$ be the partial degrees of f. Let $d = \max_i \deg_{x_i} f$, and $D = \deg\ f$. Let p_n be the n'th prime. Let

$$v_j = f(2^j, 3^j, 5^j, \ldots, p_n^j)\ for\ j = 0, 1, \ldots, 2t-1.$$

Algorithm 1 Ben-Or and Tiwari:

Input: Black box $f = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$.

Output: Monomials $M_1, M_2, \ldots, M_t$, and coefficients $a_1, a_2, \ldots, a_t$.

1: Let $v_j = f(2^j, 3^j, 5^j, \ldots, p_n^j)\ for\ j = 0, 1, \ldots, 2t-1$.
2: Compute $\lambda(z) = \prod_{i=1}^{t}(z - m_i)$, from the evaluations v_j.
3: Compute the roots of $\lambda(z)$.
4: Factor the roots m_i with trial division by $2, 3, ..., p_n$ to obtain M_i.
For example, if $m_i = 720 = 6! = 2^4 2^2 5$ then $M_i = x_1^4 x_2^2 x_3^1$.
5: Solve the following Vandermonde system for the coefficients $a_1, a_2, ..., a_t$.

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{bmatrix}$$

.

The modular implementation of the Ben-Or and Tiwari algorithm runs steps 1,2, and 4, mod p, where p is a prime. To ensure that the monomials mod p are unique it is required that $p > m_i \leq 2^{d_1} 3^{d_2} \ldots p_n^{d_n} \leq p_n^D$, and therefore the size of the prime needed is $O(D \log p_n)$. The algorithm requires $2t \in O(t)$ evaluations if t is known. In their presentation, the authors assumed a term bound T is given. In most applications a good term bound is not readily available. Kaltofen shows in [6] how to modify steps 1 and 2 to determine t with high probability using a few additional evaluation points. In this thesis we will assume t has been determined.

2.1.3 The Ben-Or and Tiwari Algorithm with Discrete Logarithms

The discrete logarithm modification of the Ben-Or and Tiwari algorithm reduces the size of the prime from $O(D \log p_n)$ to $O(n \log d)$. The method involves an application of Fermat's little theorem and the computation of a discrete logarithm.

Definition 2.1.1. Let $G = \langle g \rangle$ be a cyclic group with order n. The **discrete logarithm problem** for the group G can be stated as follows.

$$Let\ h \in G.\ Given\ G = \langle g \rangle, find\ an\ integer\ a\ such\ that\ g^a = h.$$

The cyclic group in our application is the prime field of units $\mathbb{Z}_p^*$. At this time, there does not exist a fast algorithm that will solve the discrete logarithm problem in general. However, if the prime p is smooth, then the Pohlig-Hellman algorithm can be used to compute the discrete logarithm efficiently. See [7].

Definition 2.1.2. A prime p is y-smooth if for every prime $q|p-1$ we have $q \leq y$.

The method requires a smooth prime $p = q_1 q_2 \dots q_n + 1$ with $q_i > \deg_{x_i} f$ and $\gcd(q_i, q_j) = 1$, and a random generator $\alpha \in \mathbb{Z}_p$. The evaluation points $(2^j, 3^j, 5^j, \dots, p_n^j)$ are modified to $(\omega_1^j, \omega_2^j, \dots, \omega_n^j)$ where $\omega_i = \alpha^{\frac{(p-1)}{q_i}}$. Notice that $\omega_i = \alpha^{\frac{(p-1)}{q_i}} \rightarrow \omega_i^{q_i} = 1$, by Fermat's little theorem. The substitution changes the roots m_i of the λ polynomial. Since $M_i = \prod_{k=1}^n x_k^{d_k}$ we now have that $m_i = \prod_{k=1}^n \omega_k^{d_k}$. The partial degrees d_k are determined by computing discrete logarithms with base α as follows:

$$\begin{aligned}\log_\alpha m_i = \log_\alpha \prod_{k=1}^{n} \omega_k^{d_k} &= \sum_{k=1}^{n} d_k \frac{p-1}{q_k} \\ &= d_1(q_2 q_3 \dots q_n) + d_2(q_1 q_3 \dots q_n) + \dots + d_n(q_1 q_2 \dots q_{n-1}).\end{aligned}$$

By taking the result modulo q_k, for each $1 \leq k \leq n$, the partial degrees d_k of M_i are determined.

The number of required evaluations does not change (i.e. $2t \in O(t)$ evaluations are required). Since $q_i > \deg_{x_i} f \leq d$, the algorithm requires that $q_i \geq d+1$ and therefore

$$\begin{aligned} p = q_1 q_2 \dots q_n + 1 &\geq (d+1)^n + 1 \\ \Rightarrow p &\geq (d+1)^n + 1 \\ \Rightarrow p &> (d+1)^n. \end{aligned}$$

Therefore the size of the prime required is $O(n \log d)$.

2.1.4 Kronecker Substitutions

Kaltofen in [1] showed that the discrete logarithms method can be modified efficiently if a Kronecker substitution is used to reduce multivariate interpolation to a univariate interpolation and a prime p of the form $2^k s + 1$ with s small is chosen.

Definition 2.1.3. Let D be an integral domain. Let $f \in D[x_1, x_2, ..., x_n], f \neq 0$. Let $r = [r_1, r_2, ..., r_{n-1}] \in \mathbb{Z}^{n-1}, r_i > d_i = \deg_{x_i} f$. Let $K_r : D[x_1, x_2, ..., x_n] \rightarrow D[x]$ be the **Kronecker substitution** $K_r(f) = f(x, x^{r_1}, x^{r_1 r_2}, ..., x^{r_1 r_2 \dots r_{n-1}})$.

Note that K_r is invertible if $r_i > d_i$, for $1 \leq i \leq n-1$.

Example 3. Let

$$f(x_1,x_2,x_3,x_4,x_5,x_6) = 2x_1x_2^3x_3^5x_4^2x_5^9x_6 + x_2^6x_3^9x_4^9x_5x_6^3 + 3x_1^9x_2^6x_3^9x_6 - 5x_1x_2^9x_4^2x_5^3x_6^5 - x_1x_2x_3x_4^3x_5^6x_6^9.$$

Each partial degree of f is 9. Therefore $r = [1, 10, 100, 1000, 10000, 100000]$ and

$$\begin{aligned} K_r(f) &= f(x, x^{10}, x^{100}, x^{1000}, x^{10000}, x^{100000}) \\ &= 2x^{192531} + x^{319960} + 3x^{100969} - 5x^{532091} - x^{963111}. \end{aligned}$$

The Kronecker substitution is applied to the multivariate target polynomial $f(x_1, ..., x_n) = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, ..., x_n]$. Let $r_i > d_i$, for all $1 \le i \le n-1$, and let

$$g(x) = K_r(f) = f(x, x^{r_1}, x^{r_1 r_2}, ..., x^{r_1 r_2 \ldots r_{n-1}}) = \sum_{i=1}^{t} a_i x^{e_i}.$$

Our goal is to develop a modular extension of the Ben-Or and Tiwari algorithm for exponent recovery. By reducing multivariate interpolation to a univariate interpolation we greatly simplify this task. To interpolate f, we first interpolate g. If $r_i > d_i$ then K_r is invertible and we can recover f easily. Note, in the black box model, to compute $g(a)$, for $a \in \mathbb{Z}_p$, we replace the input

$$(a_1, a_2, \ldots, a_n) \in \mathbb{Z}_p^n$$

with

$$(a, a^{r_1}, a^{r_1 r_2}, \ldots, a^{r_1 r_2 \ldots r_{n-1}}) \in \mathbb{Z}_p^n.$$

To determine $g = K_r(f)$, the prime p must still be larger than the monomials m_i. How large could the monomials be? Lets imagine a worst case scenario with monomial $x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}$ and determine a bound for p. Assuming $d = \max d_i$,

$$\begin{aligned} &K_r(x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}) = x^{d + d(d+1) + \ldots + d(d+1)^{n-1}} \\ &= x^{d(\frac{(d+1)^n - 1}{d})} = x^{(d+1)^n - 1} \\ &\Rightarrow \log_\alpha \alpha^{(d+1)^n - 1} = (d+1)^n - 1. \end{aligned}$$

Therefore, $p > (d+1)^n$ suffices and the size of the prime needed is $O(n \log d)$ which is the same as before the Kronecker substitution is applied.

The Kronecker substitution also simplifies the main algorithm. The evaluations are simplified to $v_j = \alpha^j$ and the roots of the λ polynomial are simplified to $m_i = \alpha^{e_i \mod p-1}$.

Therefore the univariate exponents are determined by computing

$$\log_\alpha(\alpha^{e_i \mod p-1}) \mod p = e_i \mod p-1.$$

2.1.5 The Generalized Chinese Remainder Theorem

If f is a polynomial in many variables or high degree, the prime $p > (d+1)^n$ will be very large. The prime p can become large enough to exceed the register size of the computing machine. In these cases arithmetic becomes expensive. The purpose of the method presented in this thesis is to reduce this expense. We present a modular extension that runs the algorithm modulo several machine primes. A difficulty that we faced was the following. Each time $\lambda(z) = \prod_{i=1}^{t}(z - \alpha^{e_i \mod p_k-1})$ is computed and factored, the roots are assigned a new order. The λ polynomial splits into linear factors and the order of the roots are defined arbitrarily. Therefore, to recover the exponents, the images mod $p-1$ first have to be assigned a correct order. Our method involves choosing primes $p_1, p_2, \ldots, p_s$ such that $p-1$ share a common divisor. We choose our primes so that this divisor is large and sort the exponents modulo this divisor. Since the moduli $p-1$ are not relatively prime, to recover the exponents, we use a generalized form of the Chinese remainder theorem which does not require relatively prime moduli. The generalized Chinese remainder theorem guarantees unique integers modulo the least common multiple of the moduli. We choose s prime so that $LCM(p_1 - 1, p_2 - 1, \ldots, p_s - 1) > (d+1)^n$. In [8] Murao and Fujise also used the generalized Chinese remainder theorem to develop a modular extension of the Ben-Or and Tiwari algorithm.

2.2 An Example

Example 4. Let $f \in \mathbb{Z}[x, y, z, w]$ be the following polynomial:

$$\begin{aligned} f(x,y,z,w) = {} & 194x^{227743}y^{443802}z^{28326}w^{17525} - 227x^{101808}y^{548145}z^{89669}w^{49883} \\ & + 423x^{226283}y^{436997}z^{100540}w^{34879}. \end{aligned}$$

After applying the Kronecker substitution we have

$$\begin{aligned} g(x) = K_r(f) &= f(x, x^{227744}, x^{124836962624}, x^{12551233059179584}) \\ &= 194x^{219963895594998967455} - 227x^{626104352821493556816} + 423x^{437787009078870598347}. \end{aligned}$$

To illustrate the method, we choose primes p so that $\Delta = 10$ is a divisor of $p - 1$. The primes we choose are the following:

$$\begin{aligned} p_1 &= 10 \cdot 110539728 + 1 \\ p_2 &= 10 \cdot 7421875 + 1 \\ p_3 &= 10 \cdot 54824434 + 1. \end{aligned}$$

Therefore we have

$$\begin{aligned} LCM(p_1 - 1, p_2 - 1, p_3 - 1) &= 22489312972688493750000 \\ &> 626104352821493556816 = \deg g. \end{aligned}$$

The exponents are first determined modulo $p - 1$ for primes p_1, p_2, and p_3. We only show these steps with p_1. The process is the same with each prime.

1. We pick $\alpha = 22 \in \mathbb{Z}_{1105397281}$ to be our generator, and evaluate $v_i = g(22^i)$, for $0 \leq i \leq 2t - 1 = 5$.

2. Next, we use these evaluations to compute the λ polynomial:

$$\lambda(z) = z^3 + 434581602z^2 + 759602492z + 574542741.$$

   The λ polynomial can be computed with the Berlekamp Massey algorithm. See [9].

3. We compute the roots of λ with Rabin's algorithm: $743370416, 462579879, 570262665$. See [10].

4. Now we take the discrete logarithm with base 22 using the Pohlig–Hellman algorithm:

$$\begin{aligned} \log_{22}(743370416) \mod p_1 &= 516130347 \\ \log_{22}(462579879) \mod p_1 &= 647786256 \\ \log_{22}(570262665) \mod p_1 &= 393234495. \end{aligned}$$

Notice that the exponents mod $\Delta = 10$ are distinct; they are $\{5, 7, 6\}$. Repeating steps 1 through 4 with primes p_2 and p_3 we obtain:

$$\begin{aligned} E_1 &= \{393234495, 516130347, 647786256\} \\ E_2 &= \{14592455, 38567097, 56056816\} \\ E_3 &= \{18108647, 95513076, 507017295\} \end{aligned}$$

Now we can pair the images E_1, E_2, and E_3 by sorting each set mod $\Delta = 10$. After pairing the images we recover the exponents with the generalized Chinese remainder theorem.

Consider the matrix in figure 2. The exponent images are listed in each row with increasing order modulo $\Delta = 10$. Notice the 10th digit in each row is sorted in increasing order. Each column contain entries that are congruent modulo $\Delta = 10$ and that corresponds to a single exponent. The arrows below each column point to an exponent recovered with the generalized Chinese remainder theorem.

Figure 2.

$$\begin{pmatrix} 393234495 & 647786256 & 516130347 \\ 14592455 & 56056816 & 38567097 \\ 507017295 & 95513076 & 18108647 \end{pmatrix} \begin{matrix} = E_1 \\ = E_2 \\ = E_3 \end{matrix}$$

$$\downarrow \qquad \downarrow \qquad \searrow$$

$$E = \{2199638955949989674 55, 626104352821493556816, 437787009078870598347\}$$

5. The final step is to determine the coefficients a_1, a_2, and a_3. To do this we solve the following transposed Vandermonde system:

$$\begin{bmatrix} 1 & 1 & 1 \\ -169167301 & -376042798 & 154252317 \\ -315717630 & -10068892 & 235190918 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 390 \\ -293393302 \\ 486206426 \end{bmatrix}.$$

The values on the right of the system are the evaluations $v_i \mod p_1$. The entries in the Vandermonde matrix on the left are the roots of λ taken to consecutive powers. Solving the system we get:

$$a_1 = 194, a_2 = 423, \ and \ a_3 = -227.$$

2.3 Modular Exponent Recovery

2.3.1 Assigning An Order To the Exponent Images

Each time $\lambda(z) = \prod_{i=1}^{t}(z - \alpha^{e_i \mod p_k - 1})$ is computed and factored, the roots are assigned a new order. The λ polynomial splits into linear factors and the roots are defined an arbitrary order. Therefore, to recover the exponents, the images first have to be assigned a corresponding order. With E denoting the set of exponents, the algorithm first computes the modular images

$$\begin{aligned} E_1 &= \{e \mod p_1 - 1 : e \in E\} \\ E_2 &= \{e \mod p_2 - 1 : e \in E\} \\ &\vdots \\ E_s &= \{e \mod p_s - 1 : e \in E\}. \end{aligned}$$

Our strategy for pairing the images is to choose primes $p_1, p_2, \ldots, p_s$ so that a common divisor Δ divides $p_k - 1$, for each $1 \leq k \leq s$. We want Δ large so the exponents are unique modulo Δ. If the exponents are unique modulo Δ, they are also unique modulo $p_k - 1$. That is, each integer in an image set E_k will be congruent modulo $p_k - 1$ to exactly one exponent. Since Δ divides $p_k - 1$, this integer is congruent modulo Δ to that exponent as well. Therefore, by sorting the images modulo Δ at the same time we pair the images with correct exponents.

For $1 \leq k \leq s$, let $e_{k1}, e_{k2}, \ldots, e_{kt}$ denote the elements of E_k. We sort the elements of E_k so that $e_{k1} \leq_\Delta e_{k2} \leq_\Delta ... \leq_\Delta e_{kt}$, where, for $x \neq y \in E$, we say $x \leq_\Delta y$ if $x \bmod \Delta \leq y \bmod \Delta$.

Figure 3.

$$\begin{array}{cccc} \left(\begin{array}{cccc} e_{11} & e_{12} & \ldots & e_{1t} \\ e_{21} & e_{22} & \ldots & e_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ e_{s1} & e_{s2} & \ldots & e_{st} \end{array}\right) & \begin{array}{l} = E_1 = E \mod p_1 - 1 \\ = E_2 = E \mod p_2 - 1 \\ \vdots \\ = E_s = E \mod p_s - 1 \end{array} \\ \begin{array}{cccc} \downarrow & \downarrow & \ldots & \downarrow \\ e_1 & e_2 & \ldots & e_t \end{array} & \begin{array}{l} \\ = E \end{array} \end{array}$$

The matrix of images sorted modulo Δ in figure 3 is a visual representation of the problem.

2.3.2 Collisions

It is possible that $e_i \equiv e_j \pmod{p_k - 1}$, for $i \neq j$. If this happens we have a collision and $|E_k| < |E|$. We call this a p–collision. However, if we choose $p_k \gg t^2$, by the birthday paradox, p–collisions are unlikely. A more serious problem is when $e_i \equiv e_j \pmod{\Delta}$, for $i \neq j$.

Definition 2.3.1. Let $\{e_1, e_2, ..., e_t\}$ be the target exponents, and Δ an integer. If $e_\rho \equiv e_\sigma \pmod{\Delta}$, for some $1 \leq \rho < \sigma \leq t$, we call this a Δ–collision.

Let X count the number of Δ–collisions.

Proposition 1. $Pr\{X = 0\} = \dfrac{\Delta!}{(\Delta - t)!\Delta^t}$.

Proof. Let $\{e_1, e_2, ..., e_t\}$ be the target exponents, and let Δ be an integer. Assume that $\{e_1, e_2, ..., e_t\}$ are random integers in $[0, (d+1)^n - 1]$ and therefore $e_i \mod \Delta$ is random mod Δ. If $X = 0$ then there are no collisions and each e_i is unique mod Δ. Therefore, the problem reduces to sampling t unique values with replacement from $\{0, 1, ..., \Delta - 1\}$. Since there are Δ possible values, there are

$$\Delta \cdot (\Delta - 1) \cdot (\Delta - 2) \ldots (\Delta - (t - 1))$$

ways all t values can be unique. Since there are Δ^t choices we have

$$Pr\{X = 0\} = \frac{\Delta \cdot (\Delta - 1) \cdot (\Delta - 2) \ldots (\Delta - (t-1))}{\Delta^t} = \frac{\Delta!}{(\Delta - t)!\Delta^t}.$$

□

Remark. If we take $\Delta \approx t^2$ then $Prob\{X = 0\} \approx .6$ by proposition 1.

Let X_{ij} be an indicator random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } e_i \text{ and } e_j \text{ have a } \Delta\text{–collision,} \\ 0 & \text{otherwise.} \end{cases}$$

Since X counts the number of Δ–collisions, $X = \sum_{i=1}^{t-1} \sum_{j=i+1}^{t} X_{ij}$.

Let A_{ij} be the event that $X_{ij} = 1$. Since X_{ij} is an indicator random variable, we have that

$$E[X_{ij}] = Pr\{A_{ij}\} = \frac{1}{\Delta}.$$

Proposition 2. $E[X] = \frac{\binom{t}{2}}{\Delta}$.

Proof. By properties of expectations and indicator random variables, we have the following:

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{t-1} \sum_{j=i+1}^{t} X_{ij}\right] \\ &= \sum_{i=1}^{t-1} \sum_{j=i+1}^{t} E[X_{ij}] \\ &= \sum_{i=1}^{t-1} \sum_{j=i+1}^{t} Pr\{A_{ij}\} \\ &= \sum_{i=1}^{t-1} \sum_{j=i+1}^{t} \frac{1}{\Delta} = \frac{\binom{t}{2}}{\Delta}. \end{aligned}$$

□

The notation used in this proof was taken from [11].

Remark. If $\Delta = t^2$ then

$$E[X] = \frac{1}{2} - \frac{1}{2t} = \frac{1}{2}\left[1 - \frac{1}{t}\right] \sim \frac{1}{2}.$$

Also, if $\Delta = 2^{(l-1)}t^2$, for $l \in \mathbb{Z}$, then

$$E[X] = \frac{1}{2^{(l-1)}t^2}\left[\frac{t^2 - t}{2}\right] = \frac{1}{2^l}(1 - \frac{1}{t}) \sim 2^{-l}.$$

Notice that we defined a Δ–collision to involve only a pair of values. It is possible to have collisions modulo Δ involving more than two exponents. However, such collisions are unlikely. For the sake of doing the analysis we assume that Δ–collisions with more than 2 integers do not occur.

2.3.3 Unique Exponents Modulo Δ

Here we present our method for exponent recovery. This method requires that the exponents are unique modulo Δ. Remember, Δ is a divisor of the moduli $p_k - 1$. We use the generalized Chinese remainder theorem to recover the exponents because it does not require relatively prime moduli.

Theorem 1 (Generalized Chinese Remainder Theorem [12])**.** *Let $m_1, m_2, ..., m_s$ be positive integers, and let $u_1, u_2, ..., u_s$ be any integers, and $M = LCM(m_1, m_2, ..., m_s)$. There exists a unique integer u such that $u \equiv u_i (\text{mod } m_i)$, for all $1 \leq i \leq s$, and $0 \leq u < M$, if and only if $u_i \equiv u_j \pmod{\gcd(m_i, m_j)}$, for all $1 \leq i < j \leq s$.*

Proof. Assume there exists an integer u, $0 \leq u < M$, such that $u \equiv u_i \pmod{m_i}$, for all $1 \leq i \leq s$. Clearly, for all $1 \leq i < j \leq s$, $u \equiv u_i \pmod{\gcd(m_i, m_j)}$ and $u \equiv u_j \pmod{\gcd(m_i, m_j)}$, and therefore $u_i \equiv u_j \pmod{\gcd(m_i, m_j)}$.

Assume that $u_i \equiv u_j \pmod{\gcd(m_i, m_j)}$, for all $1 \leq i < j \leq s$, and let $M_j = LCM(m_1, m_2, \ldots, m_j)$. Now, we need to determine

$$u = x_1 + M_1 x_2 + ... + M_{s-1} x_s, where\ 0 \leq x_i < \frac{M_i}{M_{i-1}}.$$

Assume that $x_1, x_2, ..., x_{j-1}$ have been determined. We can solve

$$x_j M_{j-1} + x_{j-1} M_{j-2} + ... + x_1 \equiv u_j \pmod{m_j}$$

for x_j as follows. Since $x_{j-1} M_{j-2} + x_{j-2} M_{j-3} + ... + x_1 \equiv u_{j-1} \pmod{m_{j-1}}$, and $u_i \equiv u_j \pmod{\gcd(m_i, m_j)}$, it follows that

$$x_{j-1} M_{j-2} + x_{j-2} M_{j-3} + ... + x_1 \equiv u_i \equiv u_j (\text{mod } \gcd(m_i, m_j)),\ for\ all\ 1 \leq i < j \leq s.$$

Therefore,

$$\begin{aligned} &LCM((m_1, m_j), (m_2, m_j), ..., (m_{j-1}, m_j)) = \gcd(LCM(m_1, ..., m_{j-1}), m_j) \\ &= \gcd(M_{j-1}, m_j)\ divides\ u_j - (x_{j-1} M_{j-2} + x_{j-2} M_{j-3} ... + x_1). \end{aligned}$$

Let $\alpha_j = u_j - (x_{j-1} M_{j-2} + x_{j-2} M_{j-3} ... + x_1)$ and $d_j = \gcd(M_{j-1}, m_j)$. Then

$$\frac{x_j M_{j-1}}{d_j} \equiv \frac{\alpha_j}{d_j} \pmod{\frac{m_j}{d_j}}$$

and $\frac{M_{j-1}}{d_j}$ (mod $\frac{m_j}{d_j}$) is invertible. The inverse can be found with Euclid's algorithm. Therefore,

$$x_j = (\frac{\alpha_j}{d_j})(\frac{M_{j-1}}{d_j})^{-1} \text{ (mod } \frac{m_j}{d_j})$$

and we have $u = x_1 + M_1 x_2 + ... + M_{s-1} x_s$ satisfying our requirements.

Now, assume that two solutions u and v exist such that $u \equiv v \equiv u_i$ (mod m_i), for $1 \leq i \leq s$. Then $m_i | u - v$ so that $LCM(m_1, m_2, \ldots, m_s) = M | u - v$ and therefore $u \equiv v \pmod{M}$. Therefore the solution is unique modulo M. □

Example 5. Let u be an integer such that $u \equiv 3 \pmod 6$ and $u \equiv 5 \pmod{10}$. Notice that $3 \equiv 5$ (mod $\gcd(6, 10) = 2$). Therefore it follows by the generalized Chinese remainder theorem that a unique integer modulo $LCM(6, 10) = 30$ exists. Notice that $15 \mod 6 = 3$ and $15 \mod 10 = 5$. Therefore $u = 15$ is a unique solution.

Now let u be an integer such that $u \equiv 4 \pmod 6$ and $u \equiv 5 \pmod{10}$. Since $4 \not\equiv 5 \pmod 2$ it follows by the generalized Chinese remainder theorem that no solution u exists.

Theorem 2. *Let $m_1 = \Delta x_1, m_2 = \Delta x_2, ..., m_s = \Delta x_s$, and $M = LCM(m_1, m_2, ..., m_s)$. Let E be a set of t non negative integers. Let $E_1, E_2, ..., E_s$ be sets such that, $E_k = \{x \bmod m_k : x \in E\}, 1 \leq k \leq s$. If the elements of E are distinct modulo Δ, then there exists exactly t integers, $u_1, u_2, ..., u_t$ such that*

(i) $0 \leq u_i < M$,

(ii) for each u_i, there exist exactly s integers $e_{1i}, e_{2i}, ..., e_{si}$ such that $e_{1i} \in E_1, e_{2i} \in E_2, ..., e_{si} \in E_s$, and $u_i \equiv e_{ki} (\text{mod } m_k)$, for all $1 \leq k \leq s$.

Proof. Since the elements of E are distinct mod Δ, they must also be distinct mod m_k, for all $1 \leq k \leq s$. If this were not the case, if $\exists\ x \neq y \in E$ such that $x \equiv y \pmod{m_k}$, then, since $\Delta | m_k$, it follows that $x \equiv y \pmod \Delta$ and this is a contradiction. Therefore, the values of E_k are distinct mod m_k and $|E_k| = t$.

For $x \neq y \in E$, we say $x <_\Delta y$ if $x \bmod \Delta < y \bmod \Delta$. Since E contains distinct elements mod Δ, we can sort E into a list of elements $e_1, e_2, ..., e_t$ such that $e_1 <_\Delta e_2 <_\Delta ... <_\Delta e_t$. Now consider each set E_k. Let $e_{k1}, e_{k2}, ..., e_{kt}$ be the elements of E_k defined as follows: $e_{k1} = e_1 \mod m_k$, $e_{k2} = e_2 \mod m_k$ ... $e_{kt} = e_t \mod m_k$. Since $\Delta | m_k$, we have that $e_1 \equiv e_{k1} \pmod \Delta$, $e_2 \equiv e_{k2} \pmod \Delta$ ... $e_t \equiv e_{kt} \pmod \Delta$. Since E contains distinct elements mod Δ, each element e_i in E is congruent mod Δ to exactly one element in E_k (namely $e_i \mod m_k$). Therefore, since $e_1 <_\Delta e_2 <_\Delta ... <_\Delta e_t$ it follows from the definition of E_k that $e_{k1} <_\Delta e_{k2} <_\Delta ... <_\Delta e_{kt}$.

We can therefore order the elements of $E_1, E_2, ..., E_s$ in a matrix where every column con-

tains entries that are equal mod Δ.

$$\begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{14} & ... & e_{1t} \\ e_{21} & e_{22} & e_{23} & e_{24} & ... & e_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ e_{s1} & e_{s2} & e_{s3} & e_{s3} & ... & e_{st} \end{pmatrix} \begin{matrix} = E_1 \\ = E_2 \\ \vdots \\ = E_s \end{matrix}$$

By definition we have that, for all $1 \leq i \leq t$, and all $1 \leq k \leq s$, $e_i \equiv e_{ki} \pmod{m_k}$. Therefore, it follows by the generalized Chinese remainder theorem that, $e_{ki} \equiv e_{li} \pmod{\gcd(m_k, m_l)}$, for $k < l$, and there exists integers $u_1, u_2, ..., u_t$ such that $0 \leq u_i < M$ and $u_i \equiv e_{ki} \pmod{m_k}$, for all $1 \leq k \leq s$.

Furthermore, The integers $u_1, u_2, ..., u_t$ are incongruent mod M. If this were not the case, if $\exists\ \rho < \sigma \in \{1, ..., t\}$, such that $u_\rho \equiv u_\sigma \pmod{M}$, then $u_\rho \equiv u_\sigma \pmod{m_k}$, for all $1 \leq k \leq s$, and thus $u_\rho \equiv u_\sigma \pmod{\Delta}, u_\rho \equiv e_{k\rho} \pmod{\Delta}$, and $u_\sigma \equiv e_{k\sigma} \pmod{\Delta}$, which implies that $e_{k\rho} \equiv e_{k\sigma} \pmod{\Delta}$, contradicting $e_{k\rho} <_\Delta e_{k\sigma}$. □

The proof of theorem 2 illustrates our method for recovering the exponents.

Algorithm 2 Generalized Chinese Remainder Algorithm: GCHREM($\vec{u}, \vec{m}$)

Input: $u_1, ..., u_n$ and $m_1, ..., m_n$, where $u_i \equiv u_j (\text{mod } gcd(m_i, m_j)), 1 \leq i < j \leq n$.

Output: u and M such that $u \equiv u_i (\text{mod} m_i)$, for all $1 \leq i \leq n$, and $0 \leq \mathbf{u} < M$.

1: **if** $n = 1$ **then**
2: **return** $u_1 \bmod m_1, m_1$.
3: **else**
4: $V, M \leftarrow GCHREM(u_1, ..., u_{n-1}, m_1, ..., m_{n-1})$.
5: Solve $sM + tm_n = \gcd(M, m_n) = d$, for t and d, using the extended euclidean algorithm.
6: $b \leftarrow (\mathrm{u}_n - V)/d$.
7: $modulus \leftarrow m_n/d$.
8: $X \leftarrow tb \bmod modulus$.
9: **return** $M \cdot X + V, M \cdot modulus$.
10: **end if**

Algorithm 3 Exponent Recovery: ER($\vec{E}$,$\vec{m}$,Δ)

Input: Sets $E_1, E_2..., E_s$ integers $m_1 = \Delta x_1, \ldots, m_s = \Delta x_s$, and Δ.

Output: E such that $E \mod m_i = E_i$, or FAIL.

1: **if** $|E_1 \mod \Delta| \neq |E_1|$ **then**
2: **return** FAIL.
3: **end if**
4: Sort $E_1, E_2, ..., E_s \bmod \Delta$.
5: **for** $i \leftarrow 1, t$ **do**
6: $x, M \leftarrow$GCHREM($[\mathrm{E}_{1i}, ..., E_{si}], [m_1, ...m_s]$).
7: $E \leftarrow \mathrm{E} \cup x$.
8: **end for**
9: **return** E.

2.3.4 Collisions Modulo Δ

In this section we present details of a possible second method for exponent recovery that permits a relatively small number of Δ–collisions. The strategy for pairing the modular images again involves sorting the images modulo Δ. We sort the elements $e_{ki} \in E_k$ so that $e_{k1} \leq_\Delta e_{k2} \leq_\Delta \ldots \leq_\Delta e_{kt}$ and order the elements in the following matrix:

$$\begin{pmatrix} e_{11} & e_{12} & e_{13} & e_{14} & ... & e_{1t} \\ e_{21} & e_{22} & e_{23} & e_{24} & ... & e_{2t} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ e_{s1} & e_{s2} & e_{s3} & e_{s3} & ... & e_{st} \end{pmatrix} \begin{matrix} = E_1 \\ = E_2 \\ \vdots \\ = E_s \end{matrix} .$$

After sorting the images modulo Δ, the entries of each column are congruent modulo Δ. However, if the exponents are not unique modulo Δ, each Δ–collision will result in a pair of adjacent columns with entries congruent modulo Δ. A correct exponent can be recovered from some combination of entries $y_1 \in E_1,\ y_2 \in E_2,\ \ldots,\ y_s \in E_s$ from adjacent columns with congruent entries. However, we don't know what the combination is. Each column that does not correspond to a Δ–collision will contain the modular images of a target exponent.

For adjacent columns corresponding to a Δ–collision, we take all combinations of entries that satisfy the generalized Chinese remainder theorem allowing for values to be constructed which do not correspond to an exponent. These superfluous values will be assigned a zero coefficient in the final stage of the extended Ben-Or/Tiwari algorithm. Let $s_1, s_2, \ldots, s_{t'}$ be the superfluous values and let $S = \{e_1, \ldots, e_t, s_1, \ldots, s_{t'}\}$ where $t' \leq t$.

Let

$$g(x) = \sum_{i=1}^{t} a_i x^{e_i} + \sum_{i=1}^{t'} b_i x^{s_i}, \ where\ b_i = 0\ for\ 1 \leq i \leq t'.$$

In the last step of the algorithm where the coefficients are determined, the following Vandermonde system will be solved for the coefficients $a_1, \ldots, a_t, b_1, \ldots, b_{t'}$:

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ldots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \ldots & m_t^{t-1} \\ m_1^t & m_2^t & \ldots & m_t^t \\ \vdots & \vdots & \ldots & \vdots \\ m_1^{t+t'-1} & m_2^{t+t'-1} & \ldots & m_t^{t+t'-1} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_t \\ b_1 \\ \vdots \\ bt' \end{bmatrix} = \begin{bmatrix} v_0 \\ \vdots \\ v_{t-1} \\ v_t \\ \vdots \\ v_{t+t'-1} \end{bmatrix}.$$

Since the matrix on the left is a transposed Vandermonde matrix we have that

$$\det M = \det \begin{bmatrix} 1 & \ldots & 1 \\ \vdots & \ldots & \vdots \\ m_1^{t+t'-1} & \ldots & m_t^{t+t'-1} \end{bmatrix} = \prod_{i>j} m_i - m_j.$$

We know that $\det M \neq 0 \Leftrightarrow m_1, \ldots, m_t$ are distinct. We know that $m_1, \ldots, m_t$ are distinct because the exponents are distinct. Therefore $a_1, \ldots, a_t, b_1, \ldots, b_{t'}$ is a unique solution. And, since

$$g(x) = \sum_{i=1}^{t} a_i x^{e_i}$$

is the true polynomial, it must be that $b_1, \ldots, b_{t'}$ are all zero.

For $m_1 = \Delta x_1, m_2 = \Delta x_2, ..., m_s = \Delta x_s$, the generalized Chinese remainder problem is to solve for an integer u, where $u \equiv e \mod m_i$, for $1 \leq i \leq s$. Notice that if $\Delta, x_1, \ldots, x_s$ are pairwise relatively prime then

$$\begin{aligned} & u \equiv e \ (\text{mod}\ m_i = \Delta x_i = LCM(\Delta x_i)),\ for\ all\ 1 \leq i \leq s, \\ \Leftrightarrow\ & u \equiv e \ (\text{mod}\ \Delta)\ and\ u \equiv e \ (\text{mod}\ x_i),\ for\ all\ 1 \leq i \leq s. \end{aligned}$$

Furthermore, if $\Delta, x_1, \ldots, x_s$ are pairwise relatively prime, then $LCM(m_1, m_2, \ldots, m_s)$ $= \Delta \cdot x_1 \cdot x_2 \cdot \ldots \cdot x_s$. Therefore, in this special case, the generalized Chinese remainder problem has an equivalent standard Chinese remainder problem. That is, we can recover the exponents with the standard Chinese remainder theorem. With $p_1 - 1 = \Delta x_1, p_2 - 1 =$

$\Delta x_2, \ldots, p_s - 1 = \Delta x_s$, the modular image sets become

$$\begin{aligned} E_\Delta &= \{e \mod \Delta : e \in E\} \\ E_1 &= \{e \mod x_1 : e \in E\} \\ &\vdots \\ E_s &= \{e \mod x_s : e \in E\}. \end{aligned}$$

Theorem 3. *Let $m_1 = \Delta x_1, m_2 = \Delta x_2, ..., m_s = \Delta x_s$, and $M = LCM(m_1, m_2, ..., m_s)$. Let E be a set of t non negative integers with $\max E < M$. Let $E_1, E_2, ..., E_s$ be sets such that $E_k = \{x \mod m_k : x \in E\}$, for $1 \leq k \leq s$. Assume that exactly two elements of E are congruent modulo Δ. Let S denote the set of integers $y_1, y_2, \ldots, y_{|S|}$ that have the following properties:*

(i) $0 \leq y_i < M$,

(ii) for each y_i, there exist exactly s integers $e_{1i}, e_{2i}, ..., e_{si}$ such that $e_{1i} \in E_1, e_{2i} \in E_2, \ldots, e_{si} \in E_s$, and $y_i \equiv e_{ki} (mod\ m_k)$, for all $1 \leq k \leq s$.

Note that S is the set of solutions. If $\Delta, x_1, x_2, \ldots, x_s$ are pairwise relatively prime and $|E_1| = |E_2| = \ldots = |E_s| = t$ then $|S| = 2^s + t - 2$.

Proof. Let a and b denote the two elements of E that are congruent modulo Δ. Consider the set $E \setminus \{a, b\}$. This set contains $t - 2$ unique elements modulo Δ. Therefore, since $\max E < M$, by theorem 2 there exists $t - 2$ elements with properties i) and ii).

Now consider $\{a, b\} \subseteq E$. Since $|E_1| = |E_2| = \ldots = |E_s| = t$, we know that $a \not\equiv b \pmod{m_k}$, for any $1 \leq k \leq s$. Since $\Delta, x_1, x_2, \ldots, x_s$ are relatively prime, for all $1 \leq k \leq s$,

$$\begin{aligned} & a \equiv b (\mod \Delta) \text{ and } a \equiv b \pmod{x_k} \\ \Leftrightarrow\ & a \equiv b \pmod{LCM(\Delta\ x_k) = \Delta x_k = m_k}. \end{aligned}$$

Since $a \equiv b \pmod{\Delta}$ and $a \not\equiv b \pmod{m_k}$ it follows that $a \not\equiv b \pmod{x_k}$ for any $1 \leq k \leq s$.

Consider the following matrix with 2 columns and $s + 1$ rows:

$$\begin{pmatrix} a & \mod \Delta = b & \mod \Delta \\ a & \mod x_1 \quad b & \mod x_1 \\ & \vdots & \vdots \\ a & \mod x_s \quad b & \mod x_s \end{pmatrix}.$$

Consider all combinations of $s+1$ integers from the matrix where 1 integer is taken from each row. Since $a \equiv b \pmod{\Delta}$ the entries in the first row are the same. Since $a \not\equiv b \pmod{x_k}$,

for any $1 \leq k \leq s$, the entries in the remaining rows are distinct. It follows that 2^s distinct combinations can be taken. Each is a combination of distinct integers. Therefore it follows by the standard Chinese remainder theorem that the integers of each combination are the residues of a unique integer modulo $\Delta \cdot x_1 \cdot \ldots \cdot x_s$.

For an integer y, since

$$\begin{aligned} & y \equiv a \pmod{\Delta} \text{ and } y \equiv a \pmod{x_k} \\ \Leftrightarrow\ & y \equiv a \pmod{m_k} \end{aligned}$$

and

$$\begin{aligned} & y \equiv b \pmod{\Delta} \text{ and } y \equiv b \pmod{x_k} \\ \Leftrightarrow\ & y \equiv b \pmod{m_k}, \end{aligned}$$

it follows that there are 2^s combination $y_1 \in E_1,\ y_2 \in E_2,\ \ldots,\ y_s \in E_s$ with $y \equiv y_i (\text{mod } m_i)$, for all $1 \leq i \leq s$, such that y is unique modulo M.

Therefore $|S| = 2^s + t - 2$. □

Remark. It follows from theorem 3 that, if $\Delta, x_1, \ldots, x_s$ are relatively prime, then for each Δ–collision $2^s - 2$ superfluous values will be constructed.

If $\Delta, x_1, x_2, \ldots, x_s$ are not relatively prime then it is often the case that $|S| < |X| \cdot (2^s - 2)$, where $|X|$ counts the number of Δ–collisions and s is the number of primes. This is because the condition of the generalized Chinese remainder theorem eliminates many possible combinations.

Example 6. With $\Delta = 10$, let

$$\begin{aligned} E_1 &= \{3887, 3919, 4657, 6328, 6866, 8056\} && = E \mod p_1 - 1 = 10 \cdot 810 \\ E_2 &= \{16, 69, 328, 407, 1456, 4487\} && = E \mod p_2 - 1 = 10 \cdot 475 \\ E_3 &= \{578, 3237, 9706, 9907, 10069, 10516\} && = E \mod p_3 - 1 = 10 \cdot 1225 \\ E_4 &= \{2427, 4347, 4866, 6188, 9229, 11626\} && = E \mod p_4 - 1 = 10 \cdot 1573 \end{aligned}$$

In this example $x_1 = 810, x_2 = 475, x_3 = 1225$, and $x_4 = 1573$. These values are not relatively prime. Notice that $\frac{810 \cdot 475 \cdot 1225 \cdot 1573}{LCM(810,475,1225,1573)} = 125000$. We can check to see if the exponents are unique modulo Δ by reducing modulo Δ as follows: $E_1 \mod \Delta = \{7, 9, 7, 8, 6, 6\}$. Also, E_1 sorted mod $\Delta = [6, 6, 7, 7, 8, 9]$. Notice the exponents have 2 Δ–collisions. Figure 4 lists the images sorted modulo $\Delta = 10$. The arrows below each block of two columns point to the corresponding integers that can be constructed.

Figure 4.

$$\begin{pmatrix} 6866 & 8056 & 3887 & 4657 & 6328 & 3919 \\ 16 & 1456 & 407 & 4487 & 328 & 69 \\ 9706 & 10516 & 3237 & 9907 & 578 & 10069 \\ \underbrace{4866} & \underbrace{11626} & \underbrace{2427} & \underbrace{4347} & 6188 & 9229 \end{pmatrix} \begin{matrix} = E_1 \\ = E_2 \\ = E_3 \\ = E_4 \end{matrix}$$

$$\downarrow$$

$$\Big\downarrow \qquad \{14356728157, 19409265157, 43201159487, 48253696487\}$$

$$\{14356728157, 19409265157, 43201159487, 48253696487\}$$

Notice that the sets in figure 4 have only 4 out of $16 = 2^4$ integers. This is because only 4 out of 16 combinations satisfy the condition of the generalized Chinese remainder theorem. This is the case for both collisions. The remaining two columns do not correspond to a collision and this is presented in figure 5.

Figure 5.

$$\begin{pmatrix} 6866 & 8056 & 3887 & 4657 & 6328 & 3919 \\ 16 & 1456 & 407 & 4487 & 328 & 69 \\ 9706 & 10516 & 3237 & 9907 & 578 & 10069 \\ 4866 & 11626 & 2427 & 4347 & \underbrace{6188} & \underbrace{9229} \end{pmatrix} \begin{matrix} = E_1 \\ = E_2 \\ = E_3 \\ = E_4 \end{matrix}$$

$$\downarrow$$

$$\Big\downarrow \quad 35319883819$$

$$2318104828$$

Remark. If $\Delta, x_1, \ldots, x_s$ are not relatively prime then $LCM(p_1 - 1, p_2 - 1, \ldots, p_s - 1)$ is smaller and therefore more primes are required to recover the exponents.

2.4 Pseudo Code

Algorithm 4 High Degree Polynomial Interpolation:

Input: Black box $f = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$.

Output: Polynomial f with monomials $M_1, M_2, \ldots, M_t$ and coefficients $a_1, a_2, \ldots, a_t$.

1: Apply $g(x) = K_r(f) = \sum_{i=1}^{t} a_i x^{e_i}$, with $r = [r_1, r_2, \ldots, r_{n-1}]$ and $r_i > \deg x_i$ f.
2: Pick s smooth primes $p_i = \Delta x_i + 1$ with $\Delta \geq t^2$ and $LCM(p_1 - 1, \ldots, p_s - 1) > (d+1)^n$, where $d = \max_i \deg_{x_i} f$.
3: **for** $k \leftarrow 1, s$ **do**
4: Pick a random generator $\alpha \in \mathbb{Z}_{p_k}$ and evaluate $v_j = g(\alpha^j)$, for $0 \leq j \leq 2t - 1$.
5: Compute $\lambda(z) = \prod_{i=1}^{t}(z - \alpha^{e_i \mod p_k - 1})$ from the evaluations v_j.
6: Compute the roots of $\lambda(z) : \alpha^{e_1 \mod p_k - 1}, \alpha^{e_2 \mod p_k - 1}, ..., \alpha^{e_t \mod p_k - 1} \in \mathbb{Z}_{pk}$.
7: Compute $\log_\alpha(\alpha^{e_i \mod p_k - 1}) \mod p_k = e_i \mod p_k - 1$ determining $E_k = \{e \mod p_k - 1 : e \textit{ is a target univariate exponent}\}$.
8: **end for**
9: **if** $|E_1 \mod \Delta| = |E_1|$ **then**
10: $E \leftarrow ER(E_1, \ldots, E_s, p_1 - 1, \ldots, p_s - 1, \Delta)$.
11: **else**
12: Set $\Delta = 2 \cdot \Delta$ or $3 \cdot \Delta$ and **goto** step 2.
13: **end if**
14: Invert the Kronecker map $= K_r^{-1}(g) = f$ determining monomials $M_1, M_2, \ldots M_t$.
15: Solve the following transposed Vandermonde system for the coefficients $a_1, a_2, ..., a_t$:

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_t \end{bmatrix}.$$

16: **Return** f.

2.5 Complexity Analysis

In this section some complexity details are given. The complexities are in terms of the number of arithmetic operations in $\mathbb{Z}_p$. Let $F(t)$ denote the cost of one evaluation of the black box polynomial f. The algorithm requires $2t \in O(t)$ evaluations and therefore the total cost of evaluations are $F(t)O(t) = O(F(t)t)$. From the evaluations, the Berlekamp Massey algorithm can be used to compute the λ polynomial in $O(t^2)$ arithmetic operations. See [9] for detials. Rabin's algorithm can be used to factor the λ polynomial in $O(t^2 \log p)$

arithmetic operations. See [10] for details. We use the Pohlig–Hellman algorithm to solve the discrete logarithm. If $p = p_1^{f_1} p_2^{f_2} \dots p_k^{f_k} + 1$ then the cost of running the Pohlig-Hellman algorithm is

$$O(\sum_{i=1}^{k} f_i(\log p + \sqrt{p_i})).$$

Details for the Pohlig–Hellman algorithm can be found in [7]. In the last step of the algorithm a Vandermonde system is solved for the coefficients. This can be done in $O(t^2)$ arithmetic operations. See [4] for details.

We summarize these costs in the following table and where it is relevant we compare with fast timings. In the table, let $M(t)$ denote the cost of multiplying two degree t polynomials in $\mathbb{Z}_p[t]$.

Step	Fast	Classical
Evaluations	NA	$O(F(t)t)$
Computing $\lambda(z)$	$O(M(t)\log t)$	$O(t^2)$
Factoring $\lambda(z)$	$O(M(t)\log t \log p)$	$O(t^2 \log p)$
Computing the discrete logarithm	NA	$O(\sum_{i=1}^{k} f_i(\log p + \sqrt{p_i}))$
Solving the Vandermonde system	$O(M(t)\log t)$	$O(t^2)$

Let $BT(t)$ denote the number of arithmetic operations in $\mathbb{Z}_p$ used in the modified Ben-Or and Tiwari algorithm. That is, $BT(t)$ denotes the sum of the costs in the chart above, which could be either fast or classical. If we need $p > (d+1)^n$ then the cost of arithmetic in $\mathbb{Z}_p$ is $O(\log^2 p)$ and therefore $O(n^2 \log^2 d)$. Assume there are s primes chosen with

$$LCM(p_1 - 1, p_2 - 1, ..., p_s - 1) > (d+1)^n$$

and assume that each prime is a machine prime. With this assumptions, arithmetic has constant cost.

Theorem 4. *The number of arithmetic operations in $\mathbb{Z}_p$ that algorithm High Degree Polynomial Interpolation does is*

$$O(s \cdot BT(t) + ts^2)$$

Proof. The cost of running High Degree Polynomial Interpolation is the number of primes s multiplied by $BT(t)$ plus the cost of applying the Chinese remainder theorem. Given s primes, the cost of using the Chinese remainder theorem to recover each individual exponent is $O(s^2)$. See [13]. Since there are t exponents the cost of recovering every exponent is $tO(s^2) \in O(ts^2)$. Therefore, the cost is $O(s \cdot BT(t) + ts^2)$. □

Chapter 3

Smooth Prime Numbers

3.1 An Algorithm for Generating Smooth Prime Numbers

The algorithm High Degree Polynomial Interpolation computes a discrete logarithm over $\mathbb{Z}_p^\star$. The computation is efficient if p is smooth and the Pohlig–Hellman algorithm is used.

Definition 3.1.1. A prime p is y-smooth if for every prime $q|p-1$ we have $q \leq y$.

In this chapter we focus on the problem of obtaining enough smooth prime numbers for the algorithm to work efficiently.

The algorithm requires s smooth primes with $LCM(p_1-1, p_2-1, \ldots, p_s-1) > (d+1)^n$, where each prime has the form $p = \Delta x + 1$ where $\Delta > t^2$ and $p \gg t^2$ so that there are few Δ–collisions and no p–collisions.

Algorithm 5 Generate Smooth Prime Numbers:

Input: $1 \leq 2\Delta \leq 2^{32}, d$, and y, where 2Δ is smooth and y is small.

Output: $y-$smooth primes $q_1, q_2, ..., q_s$ such that $\Delta | q_i - 1$ and $LCM(q_1 - 1, \ldots q_s - 1) > d$, or FAIL.

$L \leftarrow 2\Delta$.
$i \leftarrow 1$.
$PRIMES \leftarrow [\ ]$.
while $L \leq d$ and $i < 169$ **do**
    Let p_i be the i'th prime.
    $i \leftarrow i + 1$.
    $k \leftarrow k$ such that $2^{60} \leq 2\Delta \cdot p_i^k < 2^{63}$.
    $n \leftarrow \lfloor \frac{2^{63}}{2\Delta \cdot p_i^k} \rfloor$.
    while $L \leq d$ and $n > 1$ **do**
        if n is y–smooth **then**
            if $2\Delta \cdot p_i^k \cdot n + 1$ is prime **then**
                if $2\Delta \cdot p_i^k \cdot n + 1 \notin PRIMES$ **then**
                    $PRIMES \leftarrow PRIMES \cup \{2\Delta \cdot p_i^k \cdot n + 1\}$.
                    $L \leftarrow LCM(L, p_i^k \cdot n)$.
                    $n \leftarrow 1$.
                end if
            end if
        end if
        $n \leftarrow n - 1$.
    end while
end while
if $L > d$ **then**
    return $PRIMES$.
else
    return $FAIL$.

3.2 Smooth Prime Estimates

In this section we discuss and present estimates for smooth prime densities.

Definition 3.2.1. An integer x is **y-smooth** if for every prime $p|x$ we have $p \leq y$. The number of y-smooth integers is

$$\Psi(x,y) = \sum_{\text{integers } m \leq x \text{ such that } m \text{ is } y-\text{smooth}} 1.$$

Example 7. We computed $\Psi(2^{30}, 1024) = 63750580$.

For $u = \frac{\log x}{\log y}$, Karl Dickman in [14] proved that

$$\Psi(x,y) \sim x\rho(u),$$

where $\rho(u)$ is the Dickman-de Bruijn ρ-function.

Definition 3.2.2. The **Dickman-de Bruijn ρ-function** is the solution to the following delay differential equation

$$u\rho(u) + \rho(u-1) = 0,$$

where $\rho(u) = 1, 0 \leq u \leq 1$.

We can solve for $\rho(u)$ as follows:

$$\begin{aligned} & u\rho(u) + \rho(u-1) = 0 \\ & \Rightarrow u\rho(u) = -\rho(u-1) \\ & \Rightarrow \rho(u) = -\frac{\rho(u-1)}{u} \\ & \Rightarrow \rho(u) = -\int_1^u \frac{\rho(v-1)}{v} dv + c. \end{aligned}$$

Notice that $\rho(1) = -0 + c$ and $\rho(1) = 1$, so we have that $c = 1$. Therefore

$$\rho(u) = \begin{cases} 1 & 0 \leq u \leq 1 \\ 1 - \int_1^u \frac{\rho(v-1)}{v} dv & u > 1 \end{cases}.$$

Notice that, for $1 \leq u \leq 2$,

$$\rho(u) = 1 - \int_1^u \frac{1}{v}\, dv = 1 - [\log u - \log 1] = 1 - \log u.$$

The Dickman function becomes increasingly complicated for higher values of u.

There is also a way to express $\rho(u)$ as an integral delay equation. We will show this, but first consider the following proposition.

Proposition 3. *Let f be continuous on $[a,b]$, and $F(x) = \int_a^x f(t)\ dt, a \leq x \leq b$. Then,*

$$\int_a^b f(t-1)\ dt = \int_a^{b-1} f(t)\ dt + \int_{a-1}^a f(t)\ dt.$$

Proof. By the fundamental theorem of calculus we have that

$$\int_a^b f(t-1)\ dt = F(b-1) - F(a-1)$$
$$= \int_a^{b-1} f(t)\ dt - \int_a^{a-1} f(t)\ dt = \int_a^{b-1} f(t)\ dt + \int_{a-1}^{a} f(t)\ dt.$$

□

Let $P(u) = \int_0^x \rho(t)dt$, for $x \geq 0$. Using proposition 3, we can express $\rho(u)$ as an integral delay equation as follows:

$$u\rho(u) = -\rho(u-1)$$
$$\Rightarrow \int_1^u t\rho(t)dt = \int_1^u [-\rho(t-1)]dt$$
$$\Rightarrow u\rho(u) - 1 \cdot \rho(1) - \int_1^u \rho(t)dt = -\int_1^u \rho(t-1)dt,$$

which follows from applying integration by parts. Now we have that

$$u\rho(u) = \int_1^u \rho(t)dt - \int_1^u \rho(t-1)dt + 1$$
$$= \int_1^u \rho(t)dt - (\int_1^{u-1} \rho(t)dt + \int_0^1 \rho(t)dt) + 1,$$

which follows by proposition 3. Now,

$$\int_1^u \rho(t)dt - (\int_1^{u-1} \rho(t)dt + \int_0^1 \rho(t)dt) + 1$$
$$= \int_1^u \rho(t)dt - (\int_1^{u-1} \rho(t)dt + (1-0)) + 1 = \int_1^u \rho(t)dt - \int_1^{u-1} \rho(t)dt$$
$$= P(u) - P(u-1) = \int_{u-1}^u \rho(t)dt,$$

which follows by the fundamental theorem of Calculus. Therefore we have that $u\rho(u) = \int_{u-1}^u \rho(t)dt$ and thus

$$\rho(t) = \frac{1}{u}\int_{u-1}^u \rho(t)dt.$$

Now we use the Dickman function to estimate smooth prime densities.

Definition 3.2.3. The number of y-smooth primes is

$$\pi(x,y) = \sum_{\text{integers } p \leq x \text{ such that } p-1 \text{ is } y-\text{smooth}} 1.$$

Example 8. We computed $\pi(2^{30}, 1024) = 4816780$.

Since Dickman proved that $\Psi(x,y) \sim x\rho(u)$, it has been conjectured that $\pi(x,y) \sim \pi(x)\rho(u)$ where $\pi(x)$ is the number of primes less than or equal to x. Our goal is to provide some evidence that enough smooth primes exist for the algorithm to work efficiently. Using the conjecture we can make the following estimates:

$$\pi(2^{63}, 1024) \approx 1.346744909 \cdot 10^{12},$$
$$\pi(2^{127}, 1024) \approx 2.670631580 \cdot 10^{21}.$$

The following theorem by Friedlander provides some additional evidence.

Theorem 5 (Friedlander J.B., 1989 [15]). *If $\alpha > \sqrt{e}/2 = 0.303...$ and $y > x^{\alpha}$ then there exists $c > 0$ such that*

$$\pi(x,y) > c\frac{x}{\log x}.$$

Theorem 5 is helpful for the following reasons. The prime number theorem states that

$$\pi(x) \sim \frac{x}{\log x}.$$

Therefore, theorem 5 tells us that, for every $\alpha > .303$, we can find a constant c such that $\pi(x,y) > c\pi(x)$. It is believed that theorem 5 holds for all $\alpha > 0$. We computed a table of these constants for $\alpha = .5,\ \alpha = .3\overline{3},\ \alpha = .25$, and $\alpha = 0.2$. Each constant in the table was computed from a density of 10^6 primes.

	$\alpha = 0.5$		$\alpha = 0.3\overline{3}$		$\alpha = 0.25$		$\alpha = 0.2$	
y	x	c	x	c	x	c	x	c
2^{16}	2^{32}	0.33759	2^{48}	0.05600	2^{64}	0.00595	2^{80}	0.000446
2^{18}	2^{36}	0.33343	2^{54}	0.05578	2^{72}	0.00559	2^{90}	0.000388
2^{20}	2^{40}	0.32862	2^{60}	0.05418	2^{80}	0.00562	2^{100}	0.000402
2^{22}	2^{44}	0.32845	2^{66}	0.05355	2^{88}	0.00555	2^{110}	0.000421
2^{24}	2^{48}	0.32661	2^{72}	0.05307	2^{96}	0.00554	2^{120}	0.000420
2^{26}	2^{52}	0.32560	2^{78}	0.05260	2^{104}	0.00545	2^{130}	0.000385
2^{28}	2^{56}	0.32463	2^{84}	0.05171	2^{112}	0.00543	2^{140}	0.000405
2^{30}	2^{60}	0.32260	2^{90}	0.05190	2^{120}	0.00526	2^{150}	0.000411

Notice that theorem 5 hold for $\alpha = .2$. Obviously as α becomes smaller the fraction of x^{α} smooth integers also becomes smaller. About .04% of primes are $x^{\alpha=.2}$ smooth.

3.3 A Polynomial GCD Application

In this section we present an application of the algorithm for computing greatest common divisors of polynomials over finite fields $\mathbb{Z}_p$. In particular, we investigate the number of smooth primes that the algorithm requires when used for this application.

Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and let $G = GCD(A, B)$. Let $\deg A = d_A$, $\deg B = d_B$, and let $d = \max\{d_A, d_B\}$. Let $LC(A)$ and $LC(B)$ denote the leading coefficients of A and B. Assume that A and B are primitive (i.e. $\gcd\{a_i\} = 1$ and $\gcd\{b_i\} = 1$). The *greatest common divisor problem* is to determine G from input polynomials A and B. Hu and Monagan in [16] have developed a sparse greatest common divisor algorithm that uses interpolation. The algorithm works by first computing $H = \Delta \times G$, and Δ, and then deriving G by computing H/Δ. The greatest common divisor algorithm uses sparse polynomial interpolation to interpolate the coefficients $h_i(x_1, ..., x_n)$ of H.

Kronecker substitutions are used to reduce multivariate polynomial GCD problems in $\mathbb{Z}[x_0, x_1, ..., x_n]$ to bivariate GCD problems in $\mathbb{Z}[x, y]$.

Definition 3.3.1. Let D be an integral domain. Let $f \in D[x_0, x_1, \ldots, x_n], f \neq 0$. Let $r = [r_1, r_2, ..., r_{n-1}] \in \mathbb{Z}^{n-1}, r_i > d_i = \deg_{x_i} f$. Let $K_r : D[x_0, x_1 \ldots, x_n] \to D[x, y]$ be the **Kronecker substitution** $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, ..., y^{r_1 r_2 \cdots r_{n-1}})$.

The problem is reduced to computing $GCD(K_r(A), K_r(B)) = K_r(G)$ from which G can be recovered by inverting the Kronecker substitution.

The algorithm is run mod p, and evaluations are taken over $\mathbb{Z}_p$.

Definition 3.3.2. Let p be a prime and let $f \in \mathbb{Z}[x, y]$. Define $\phi_p : \mathbb{Z}[x, y] \to \mathbb{Z}_p[x, y]$ to be the morphism $\phi_p(f) = f \mod p$.

Not all primes can be used to derive G. Consider the following example.

Example 9. Let $\overline{A} = x_0^2 + x_1^2, \overline{B} = x_0^2 + x_1^2 + p$, and $G = x_0 + x_1$. Thus, we have that $K_r(\overline{A}) = x^2 + y^2$, $K_r(\overline{B}) = x^2 + y^2 + p$, and $K_r(G) = x + y$.

$$\begin{aligned} &GCD(\phi_p(K_r(\overline{A}))\ \ \phi_p(K_r(\overline{B}))) \\ &= GCD(\phi_p(x^2 + y^2),\ \ \phi_p(x^2 + y^2 + p)) \\ &= GCD(x^2 + y^2,\ \ x^2 + y^2 + 0) = x^2 + y^2. \end{aligned}$$

Since $GCD(\phi_p(K_r(\overline{A})),\ \ \phi_p(K_r(\overline{B}))) \neq 1$, this image cannot be used to recover G.

Definition 3.3.3. Let p be a prime. If $\deg_x GCD(\phi_p(K_r(\overline{A})),\ \ \phi_p(K_r(\overline{B}))) > 0$ then we say that p is **unlucky**.

We now proceed to determine a bound on the number of unlucky primes. In doing this we use properties of the Sylvester resultant.

Definition 3.3.4. Let D be an integral domain, and let

$$f = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0 \in D[x],\ a_0 \neq 0,$$
$$g = b_n x^n + b_{n-1}x^{n-1} + \ldots + b_0 \in D[x],\ b_0 \neq 0.$$

The **Sylvester matrix** of f and g, denoted $Syl(f,g)$, is the following $(n+m)\times(n+m)$ matrix:

$$Syl(f,g) = \begin{pmatrix}
a_0 & & & & b_0 & & & \\
a_1 & a_0 & & & b_1 & b_0 & & \\
a_2 & a_1 & \ddots & & b_2 & b_1 & \ddots & \\
\vdots & & \ddots & a_0 & \vdots & & \ddots & b_0 \\
 & \vdots & & a_1 & & \vdots & & b_1 \\
a_1 & & & & b_m & & & \\
 & a_n & & \vdots & & b_m & & \vdots \\
 & & \ddots & & & & \ddots & \\
 & & & a_n & & & & b_m
\end{pmatrix}$$

The empty spaces are filled with zeros. The **Sylvester resultant** of f and g with respect to x, denoted $res_x(f,g)$, is the determinant of the Sylvester matrix. The Sylvester resultant is a polynomial with coefficients of f and g. See [17].

If $LC_x(K_r(\overline{A}))LC_x(K_r(\overline{B}))\mod p \neq 0$ then

$$\begin{aligned}
&\deg_x GCD(\phi_p(K_r(\overline{A})),\phi_p(K_r(\overline{B}))) > 0 \\
\Leftrightarrow\ &res_x(\phi_p(K_r(\overline{A})),\phi_p(K_r(\overline{B}))) = 0 \\
&and\ res_x(\phi_p(K_r(\overline{A})),\phi_p(K_r(\overline{B}))) = \phi_p(res_x(K_r(\overline{A}),K_r(\overline{B}))).
\end{aligned}$$

Therefore, p is unlucky if and only if $p|res_x(K_r(\overline{A}),K_r(\overline{B}))$. See [17] for more details. Thus, we need a bound on the height of $res_x(K_r(\overline{A}),K_r(\overline{B}))$ to bound the maximum number of unlucky primes. However, $\overline{A}$ and $\overline{B}$ are unknown. We know A and B and we know that $\overline{A}|A$ and $\overline{B}|B$.

Let S be the Sylvester matrix of $K_r(\overline{A})$ and $K_r(\overline{B})$, and $R = res_x(K_r(\overline{A}),K_r(\overline{B}))$. We have that $\deg_{x_0}\overline{A} \leq \deg_{x_0} A \leq d$ and $\deg_{x_0}\overline{B} \leq \deg_{x_0} B \leq d$ so that the dimensions of S are no larger than $2d \times 2d$. Let $m = 2d$. We use the following theorem from Goldstein and Graham to determine a bound on the height of $res_x(K_r(\overline{A}),K_r(\overline{B}))$.

Lemma 1 (Goldstein and Graham, 1974 [18]). *Let A be an $n \times n$ matrix of polynomials in $\mathbb{Z}[x]$. Let B be the matrix of one norms of the entries in A. That is, let $B_{ij} = ||A_{ij}||_1$. Let H be the Hadamard's bound for the determinant of B. Then $||\det A||_\infty \leq H$.*

Let A' be the matrix of one norms of the entries of the matrix S. Let $h = \max\{||A||_\infty, ||B||_\infty\}$. We can determine h because A and B are inputs to the GCD algorithm. We can use the following theorem to determine a bound on the height of $\overline{A}$ and $\overline{B}$.

Theorem 6 (A.O. Gelfond [19]). *Suppose $P_1(x_1, x_2, ..., x_s), ..., P_m(x_1, x_2, ..., x_s)$ are arbitrary polynomials in s variables with heights $H_1, H_2, ..., H_m$. Denoting the height and degrees of the polynomial $P(x_1, x_2, ..., x_s) = P_1(x_1, ..., x_s)...P_m(x_1, ..., x_s)$ by H and $n_1, n_2, ..., n_s$ in the variables $x_1, x_2, ..., x_s$, respectively, we will have the inequality*

$$H \geq e^{-n} H_1 H_2 \dots H_m, \ n = \sum_{i=1}^{s} n_i.$$

We have by theorem 6 that $\max\{||\overline{A}||_\infty, ||\overline{B}||_\infty\} \leq e^{(n+1)d} h = H$, where we let H denote this bound. The entries of S are polynomials $S_{ij} = \sum_{k=1}^{t_{ij}} \sigma_i y^i$, where $|\sigma_i| < e^{(n+1)d} h = H$. The maximum possible number of terms that each entire S_{ij} could have is $(d+1)^n$ which is the maximum degree after applying the Kronecker substitution. Let

$$||S_{ij}||_1 < (d+1)^n H = \Gamma, \ \textit{where } d = \max\{d_A, d_B\}.$$

Finally, we can apply Goldstein and Graham's theorem to find a bound on the height of R as follows:

$$||R||_\infty \leq H(A') = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} ||S_{ij}||^2} \leq \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} \Gamma^2} = \prod_{i=1}^{m} \sqrt{m\Gamma^2} = m^{\frac{m}{2}} \Gamma^m = m^{\frac{m}{2}} (d+1)^{nm} H^m.$$

Now lets consider a very large GCD problem. Let $h = 2^{64}, m = 2d = 40$, and $n = 8$. With these numbers we can determine H. After determining H we can determine Γ. We have that $||R||_\infty \leq m^{\frac{m}{2}} \Gamma^m$ which is a very large number. We can determine a bound on the number of unlucky 63 bit primes by calculating $\lceil \log_{2^{63}} m^{\frac{m}{2}} \Gamma^m \rceil = 230$. Therefore, this problem can have no more than 230 unlucky primes.

Chapter 4

Conclusions

Zippel's sparse interpolation algorithm is currently used in Maple, Magma, and Mathematica. The modified Ben-Or and Tiwari algorithm requires fewer evaluations than Zippel's algorithm. This is especially true for sparse polynomials. However, for polynomials with high degree or many variables the size of the prime required for the modified Ben-Or and Tiwari algorithm is large and may exceed the computing machines register size. This makes arithmetic expensive. The modular extension of the Ben-Or and Tiwari algorithm for exponent recovery presented in this thesis will efficiently handle polynomials in many variables or high degree. The algorithm can also be implemented in parallel where the exponent images modulo m_k are computed in parallel. Wang and Weber in [20] investigated the parallelism of Zippel's algorithm and found that it has limitations.

The algorithm presented in this thesis requires s primes p_k such that $p_k - 1$ share a common divisor Δ. This common divisor Δ takes up approximately half of the prime. Consequentially more primes are required to recover exponents. There may be another method that requires fewer primes.

Bibliography

[1] E. Kaltofen. Fifteen years after dsc and wlss2. *Proc. of PASCO*, pages 10–17, 2010.

[2] Go Soo. Sparse polynomial interpolation and the fast euclidean algorithm – masters thesis. *Simon Fraser University*, 2012.

[3] R. Zippel. Probabilistic algorithms for sparse polynomials. *In Proc. of EUROSAM*, 79:216–226, 1979.

[4] R. Zippel. Interpolating polynomials from their values. *J.Symb. Comput.*, 9(3):375–403, 1990.

[5] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *In Proc. of the twentieth anual ACM symposium on Theory of computing*, pages 301–309, 1988.

[6] W.Lee E. Kaltofen and A.A. Lobo. Early termination in ben-or/tiwari sparse interpolation and a hybrid of zippel's algorithm. *Proc. of ISSAC 00, ACM Press*, pages 192–201, 2000.

[7] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24, 1, 1978.

[8] Hirokazu Murao and Tetsuro Fujise. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *J. Symb. C*, 21:377–396, 1996.

[9] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Trans. on Information Theory*, 15:122–127, 1969.

[10] Michael Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput*, 9:273–280, 1979.

[11] R. Rivest C. Stein T. Cormen, C. Leiserson. Introduction to algorithms, 3rd edition. *The MIT Press Cambridge, Massachusetts*, pages 118–120, and 130–133, 2009.

[12] D. Knuth. The art of computer programming - semi-numerical algorithm. *IEEE Transactions on Information Theory*, 2, 1981.

[13] George Labahn. Keith O. Geddes, Stephen R. Czapor. Algorithms for computer algebra. *Kluwer Academic Publishers*, 1992.

[14] K. Dickman. On the frequency of numbers containing prime factors of a certain relative magnitude. *Ark. Mat. Astr. Fys.*, 21:1–14, 1930.

[15] J. B. Friedlander. Shifted primes without large prime factors. *Number Theory and Applications Banff AB*, pages 393–401, 1988.

[16] J. Hu and M. Monagan. A fast parallel sparse polynomial gcd algorithm. *Proc. ISSAC, ACM Press*, pages 271–278, 2016.

[17] D. O'Shea. D. Cox, J.Little. Ideals, varieties and algorithms. *Springer Verlag*, pages 162–168, 1991.

[18] A. Goldstein and G. Graham. A hadamard type bound on the coefficients of a determinant of polynomials. *SIAM Review*, 21:394–395, 1974.

[19] A. O. Gelfond. Translated by Leo F. Boron. Transcendental and algebraic numbers. *Dover Publications, Mineola, New York*, 21:135–139, 1960.

[20] Kenneth Weber. Mohamed Rayes, Paul Wang. Parallelization of the sparse modular gcd algorithm for multivariate polynomials on shared memory multiprocessors. *Proc. of ISSAC, ACM Press*, 94:66–73, 1994.