SOLVING LINEAR SYSTEMS OF EQUATIONS OVER CYCLOTOMIC FIELDS

by

Liang Chen

B.Sc. Simon Fraser University, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
2007

APPROVAL

Name: Liang Chen

Degree: Master of Science

Title of thesis: Solving Linear Systems of Equations over Cyclotomic Fields

Examining Committee:

Chair

Dr. Michael Monagan, Senior Supervisor
Associate Professor, Mathematics
Simon Fraser University

Dr. Petra Berenbrink, Supervisor
Assistant Professor, Computing Science
Simon Fraser University

Dr. Nils Bruin, SFU Examiner
Assistant Professor, Mathematics
Simon Fraser University

Date Approved: _____

Abstract

Let $A \in \mathbb{Q}[z]^{n \times n}$ be a matrix of polynomials and $b \in \mathbb{Q}[z]^n$ be a vector of polynomials. Let $m(z) = \Phi_k[z]$ be the k^{th} cyclotomic polynomial. We want to find the solution vector $x \in \mathbb{Q}[z]^n$ such that the equation $Ax \equiv b \bmod m(z)$ holds. One may obtain x using Gaussian elimination, however, it is inefficient because of the large rational numbers that appear in the coefficients of the polynomials in the matrix during the elimination. In this thesis, we present two modular algorithms namely, Chinese remaindering and linear p-adic lifting. We have implemented both algorithms in Maple and have determined the time complexity of both algorithms. We present timing comparison tables on two sets of data, firstly, systems with random generated coefficients and secondly real systems given to us by Vahid Dabbaghian which arise from computational group theory. The results show that both of our algorithms are much faster than Gaussian elimination.

Keywords: modular algorithm; cyclotomic field; Chinese remaindering; p-adic lifting; rational reconstruction

To my parents.

"Behind every argument is someone's ignorance."

— Louis D. Brandeis *(1856 - 1941)*

Acknowledgments

I would like to thank my parents who have always supported me in my studies, and especially their support in the last 6 years of my studies in Canada. I cannot think of any other ways to show my appreciation to them except trying my best in my studies.

I would also like to thank Dr. Michael Monagan, my senior supervisor, who brought me into this research area. I met Dr. Monagan in September 2004 in the cryptography course he was teaching. I was interested in this course because Dr. Xiaoyun Wang demonstrated collision attacks against MD5, SHA-0 and some other related hash functions in August 2004 which was considered a big achievement in cryptography. By coincidence, Michael was offering a course in cryptography in September 2004. Therefore, I joined his class and explored the world of cryptography. In the following semester, I took the computer algebra course with Michael as one of his graduate students, and then started to do research in computer algebra with him. Michael made a great effort helping me work through my research. I appreciate his patience and guidance.

My co-supervisor, Dr. Petra Berenbrink, is from the School of Computing Science who does research in probabilistic methods, randomized algorithms, and parallel computing. She has a very strong background in theory and I appreciate her guidance.

At last, I would like to thank my friends Simon, Al, Greg, and everyone in the CECM lab. We had a great time together, and I really enjoyed being with you guys!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Suppose we are given an $n \times n$ matrix A and a vector b over the rationals, and suppose we want to find a vector $x \in \mathbb{Q}^n$ such that the equation $Ax = b$ holds. One well know algorithm is Gaussian Elimination. In computer algebra, modular algorithms have been developed to speed this up.

In this thesis, we investigate algorithms for solving linear systems involving roots of unity. We were motivated to do so when Dr. Vahid Dabbaghian-Abdoly gave us a sequence of linear systems over cyclotomic fields to solve arising from a computational group theory problem. For example, one of the linear systems given to us is the following. For $\beta = \frac{-1+\sqrt{3}i}{2}$, the third root of unity,

$$A^{196\times 196} = \begin{bmatrix} \frac{109}{91}\beta - \frac{121}{182}\beta^2 & \frac{545}{182}\beta - \frac{549}{182}\beta^2 & \dots \\ \frac{423}{182}\beta + \frac{239}{182}\beta^2 & \frac{109}{182}\beta + \frac{41}{182}\beta^2 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}, b^{196} = \begin{bmatrix} 0 \\ -1 \\ \vdots \end{bmatrix},$$

Maple's `LinearAlgebra` package `LinearSolve` command, which uses Gaussian elimination to solve this linear system over algebraic number fields, obtains the answer

$$x = \begin{bmatrix} -\frac{193028420497557977963092944211837 3}{837637134068527924278537117122 85} + \frac{293530015437001131689173724428409}{167527426813705584855707423424570}\beta \\ \frac{125712863214341440313988741186775 91}{234538397539187818797990392794398 0} + \frac{170534906127849498440359300473108931}{234538397539187818797990392794398 0}\beta \\ \vdots \end{bmatrix}$$

However, it's inefficient to do it this way since it involves $O(n^3)$ multiplications and divisions of polynomials whose coefficients have large fractions. We can see from above example that the coefficients in the solution vector x are much larger than those in the input. However,

they are a lot smaller than the maximum possible coefficient if the coefficients in input matrix A and vector b were of the same size as in the example, i.e., $< 10^3$, but generated randomly. We should design the algorithms in such a way that the work they do is less if the output is small. For such random input, we obtain the maximum coefficient in the solution vector x to be about 9452 bits long by experiment, and it takes a "long time" (see Tables 2.1, 2.2, 2.3) for Gaussian elimination to obtain the solution of such a system. In Chapter 2, we develop two efficient modular algorithms namely Chinese remaindering and linear p-adic lifting. Both of the algorithms need to use rational number reconstruction to recover the rational coefficients in the solution vector as in the above example.

1.1 Cyclotomic Fields and Cyclotomic Polynomials

Definition 1.1.1 (Primitive n^{th} root of unity)**.** A complex number z which satisfies $z^n = 1$ $(n = 1, 2, 3, ...)$ is called an n^{th} root of unity. If also $z^i \neq 1$ for $1 < i < n$, then z is a *primitive* n^{th} root of unity.

Example 1.1.2. The roots of $x^3 - 1$ are 1, $-\frac{1}{2} \pm \frac{\sqrt{3}}{2}i$. The primitive 3^{rd} root of unity are $-\frac{1}{2} \pm \frac{\sqrt{3}}{2}i$.

Definition 1.1.3 (Minimal polynomial)**.** The minimal polynomial of an algebraic number $e \in \mathbb{C}$ is the monic irreducible polynomial $p(z) \in \mathbb{Q}[z]$, where $p(e) = 0$.

Example 1.1.4. The minimal polynomial for $\pm i = \pm\sqrt{-1}$ is $z^2 + 1$.

Definition 1.1.5 (Cyclotomic polynomial)**.** The zeros of the polynomial $p(z) = z^n - 1$ are precisely the n^{th} roots of unity, each with multiplicity 1. The n^{th} cyclotomic polynomial is defined by the fact that its zeros are precisely the primitive n^{th} roots of unity, each with multiplicity 1:

$$\Phi_n(z) = \prod_{k=1}^{\varphi(n)} (z - \beta_k)$$

where $z_1, ..., z_{\varphi(n)}$ are the primitive n^{th} roots of unity, and $\varphi(n)$ is the Euler's totient function.

Example 1.1.6. Table 1.1 shows some cyclotomic polynomials along with some of their roots. Note: $m(z) = \Phi_k(z)$ is the minimal polynomial for $\beta \in \mathbb{C}$.

k	$\Phi_k(z)$	β
1	$z-1$	1
2	$z+1$	-1
3	z^2+z+1	$\frac{-1\pm\sqrt{3}i}{2}$
4	z^2+1	i
5	$z^4+z^3+z^2+z+1$	$0.309+0.951i$
6	z^2-z+1	$\frac{1\pm\sqrt{3}i}{2}$
7	$z^6+z^5+z^4+z^3+z^2+z+1$	$\cos(\frac{2}{7}\pi)+\sin(\frac{2}{7}\pi)i$
8	z^4+1	$\frac{\sqrt{2}+\sqrt{2}i}{2}$
9	z^6+z^3+1	$0.766+0.642i$
10	$z^4-z^3+z^2-z+1$	$0.809+0.587i$

Table 1.1: Examples of the first ten cyclotomic polynomials. β are some of the corresponding complex root(s)

1.2 Polynomial Interpolation

One of the tools that we use in this thesis is polynomial interpolation.

Theorem 1.2.1 (Polynomial Interpolation)**.** Given a set of $n+1$ data points $(x_i, y_i) \in F^2$, F is a field, where no two x_i's are the same, there exists a unique polynomial $p(x) \in F[x]$ of degree at most n satisfying that $p(x_i) = y_i$, for $i = 0, ..., n$.

Proof. See Geddes [11] Chapter 5. □

The Algorithm We Use to Compute $p(x)$

We express the solution $p(x) \in F[x]$ in the mixed radix representation, which is also called the Newton form:

$$p(x) = v_0 + v_1(x-\alpha_0) + v_2(x-\alpha_0)(x-\alpha_1) + \cdots + v_n \prod_{i=0}^{n-1}(x-\alpha_i)$$

where the coefficients $v_i \in F$, $0 \le i \le n$, are to be determined. We can see that $p(\alpha_0) = v_0 \Rightarrow v_0 = y_0$, $p(\alpha_1) = v_0 + v_1(\alpha_1 - \alpha_0) \Rightarrow v_1 = \frac{y_1 - v_0}{\alpha_1 - \alpha_0}$, ..., etc. Therefore, we use the formula

$$v_k = \frac{y_k - [v_0 + v_1(\alpha_k - \alpha_0) + \cdots + v_{k-1}\prod_{i=0}^{k-2}(\alpha_k - \alpha_i)]}{(\alpha_k - \alpha_0)(\alpha_k - \alpha_1)\cdots(\alpha_k - \alpha_{k-1})}$$

to calculate v_k for $0 \leq k \leq n$ to obtain $p(x)$. This interpolation method takes $O(n^2)$ arithmetic operations in F to obtain $p(x)$ where n is the number of pairs of evaluation points.

Example 1.2.2. Given three points $(-5, 3), (0, 8), (10, -2)$, we want to find the quadratic polynomial $p(x) \in \mathbb{Q}[x]$ such that $p(-5) = 3$, $p(0) = 8$, and $p(10) = -2$.
Using the Newton's interpolation we have the polynomial $p(x)$ in the form:

$$p(x) = v_0 + v_1(x + 5) + v_2(x + 5)(x - 0),$$

and we would like to solve for v_0, v_1, and v_2. We know $p(-5) = v_0 = 3$, therefore $v_0 = 3$; $p(0) = 3 + v_1(0 + 5) = 8$, therefore $v_1 = 1$; $p(10) = 3 + 1(10 + 5) + v_2(10 + 5)(10 - 0) = -2$, therefore $v_2 = -\frac{2}{15}$. Hence we obtain

$$p(x) = 3 + 1(x + 5) - \frac{2}{15}(x + 5)(x - 0) = -\frac{2}{15}x^2 + \frac{1}{3}x + 8.$$

Remark: Maple uses Newton's polynomial interpolation which takes $O(n^2)$ operations in F for n pairs of evaluation points of constant lengths. One may use Lagrange interpolation instead which also takes $O(n^2)$ operations in F to compute.

1.3 Chinese Remaindering

We realize that computing with single-precision integers is considerably more efficient than computing with multiprecision integers. Therefore, we may transform a computation involving large integers into a computation with integers that can be fit into one computer word, and then recover the multiprecision integers in the solution. In this section, we discuss the Chinese remainder theorem and Chinese remainder algorithm which recovers multiprecision integers from a sequence of single-precision integers.

Theorem 1.3.1 (Chinese Remainder Theorem)**.** Let $m_0, m_1, ..., m_n \in \mathbb{Z}$ be integers which are pairwise relatively prime and let $u_i \in \mathbb{Z}$, $i = 0, 1, ..., n$ be $n + 1$ specified residues. For any fixed integer $a \in \mathbb{Z}$ there exists a unique integer $u \in \mathbb{Z}$ which satisfies the following conditions:

$$u \equiv u_i \pmod{m_i},\ 0 \leq i \leq n., \text{ and } a \leq u < a + m, \text{ where } m = \prod_{i=0}^{n} m_i. \tag{1.1}$$

Proof. See Geddes [11] Chapter 5. □

Chinese Remainder Algorithm

The algorithm which is generally used to solve the Chinese remaindering problem is named after H. L. Garner. Given positive moduli $m_i \in \mathbb{Z}$ $(0 \leq i \leq n)$ which are pairwise relatively prime and given corresponding residues $u_i \in \mathbb{Z}_{m_i}$ $(0 \leq i \leq n)$, we wish to compute the unique $u \in \mathbb{Z}_m$, where $m = \prod_{i=0}^{n} m_i$, which satisfies the system of congruences (1.1). The key to Garner's algorithm is to express the solution $u \in \mathbb{Z}_m$ in the mixed radix representation

$$u = v_0 + v_1(m_0) + v_2(m_0 m_1) + \cdots + v_n(\prod_{i=0}^{n-1} m_i)$$

where $v_k \in \mathbb{Z}_{m_k}$ for $k = 0, 1, \ldots, n$.

Theorem 1.3.1 along with the example shows that it is possible to recover the integer from it's images given enough sets of images and primes. The cost of it is analyzed in section 2.2.4. The same technique can be applied to a sequence of polynomial images by applying the CRT to each set of coefficients separately. We illustrate with and example.

Example 1.3.2. Let $p_1 = 2z^3 + 3z^2 + 4z + 5 \pmod 7$, $p_2 = 6z^3 + 7z^2 + 8z + 9 \pmod{11}$, $p_3 = 7z^3 + 8z^2 + 9z + 10 \pmod{13}$. We would like to find polynomial $p \in \mathbb{Z}[z]$ such that $p \equiv p_1 \pmod 7$, $p \equiv p_2 \pmod{11}$, and $p \equiv p_3 \pmod{13}$. Let $p = c_1 z^3 + c_2 z^2 + c_3 z + c_4$. We execute the Chinese remainder algorithm on the following four problems:

$$\{c_1 \equiv 2 \pmod 7,\ c_1 \equiv 6 \pmod{11},\ c_1 \equiv 7 \pmod{13},\}$$

$$\{c_2 \equiv 3 \pmod 7,\ c_2 \equiv 7 \pmod{11},\ c_2 \equiv 8 \pmod{13},\}$$

$$\{c_3 \equiv 4 \pmod 7,\ c_3 \equiv 8 \pmod{11},\ c_3 \equiv 9 \pmod{13},\}$$

$$\{c_4 \equiv 5 \pmod 7,\ c_4 \equiv 9 \pmod{11},\ c_4 \equiv 10 \pmod{13},\}$$

we obtain the answers $c_1 = 72$, $c_2 = 73$, $c_3 = 74$, and $c_4 = 75$. Therefore, we can write $p = c_1 z^3 + c_2 z^2 + c_3 z + c_4 = 72x^3 + 73x^2 + 74x + 75$. It is easy to verify that p is our desired polynomial satisfying all of our requirements that $p \equiv p_1 \pmod 7$, $p \equiv p_2 \pmod{11}$, and $p \equiv p_3 \pmod{13}$.

The Chinese remainder algorithm can also be applied to solve linear systems of equations over $\mathbb{Q}$. Let $A = [A_1|A_2|\cdots|A_n]$ where A_i is the i^{th} column of A and let $A^{(i)} =$

$[A_1|\cdots|A_{i-1}|b|A_{i+1}|\cdots|A_n]$ for $1 \le i \le n$. By Cramer's rule, the i^{th} entry of the solution $x \in \mathbb{Q}^n$ of $Ax = b$ is given by

$$x_i = \frac{\det(A^{(i)})}{\det(A)} \text{ for } 1 \le i \le n.$$

Here $\det(A)$ and $\det(A^{(i)})$ are integers because A, $A^{(i)}$ are matrices of integers. We can compute $\det(A)$ and $\det(A^{(i)})$ using Chinese remaindering as follows. For primes $p_1, p_2, \ldots, p_L$ such that

$$\prod p_i > 2\max(|\det(A)|, |\det(A^{(1)})|, \ldots, |\det(A^{(n)})|),$$

we solve $Ax^{(j)} = b \bmod p_j$ for $x^{(j)} \in \mathbb{Z}_{p_j}^n$ using Gaussian elimination, and at the same time we compute $d_j = \det(A) \bmod p_j$ using the fact that the determinant of a triangular matrix is the product of the diagonal entries. For each prime p, this costs $O(n^3)$ arithmetic operations in $\mathbb{Z}_p$. Now we can obtain $\det(A)$ from $d_j \bmod p_j$ by the CRT. Noting from Cramer's rule that

$$\det(A^{(i)}) = \det(A)x_i,$$

then

$$\det(A^{(i)}) \equiv \det(A)x_i \equiv d_j x_i^{(j)} \pmod{p_j}.$$

Hence, we obtain $\det(A^{(i)})$ from $(d_j x_i^{(j)} \bmod p_j, p_j)$ using Chinese remaindering. If L is the number of primes needed, the cost is $O(n^2cL + n^3L + nL^2)$ which is the cost of reducing A modulo L primes, Gaussian elimination and Chinese remaindering, where c is the length of the longest entry in A.

Remark: We use the symmetric range for $\mathbb{Z}_p$ so that we can recover negative integers. That's why we have a factor of 2 in the inequality above.

Definition 1.3.3 (Machine prime)**.** The primes which in binary format can fit into one machine word.

Example 1.3.4. The largest machine prime on a 32-bit machine is 4294967291, and 18446744073709551557 on a 64-bit machine.

Remark: Maple's `LinearAlgebra` package uses 32 bit machine primes on a 64-bit machine and 16 bit primes on a 32-bit machine. The largest machine prime that `LinearAlgebra` package supports fast arithmetic is 4294967291 on a 64-bit machine and 65521 on a 32-bit machine which is a fairly small number. In section 2.2.5, we will discuss the "run out of prime" problem where 25 bit floating point primes are suggested on a 32-bit machine.

1.4 Rational Number Reconstruction

Section 1.3 shows us it is possible to use the Chinese remainder algorithm to solve linear systems of equations over $\mathbb{Q}$, and reduce the computation into modulo operations. This method is not good if the rationals in x are small in size compare to $\det(A)$. One may also use an output sensitive Chinese remainder algorithm to solve linear systems $Ax = b$ over $\mathbb{Q}$ with rational number reconstruction. Rational reconstruction was invented by Paul Wang in [3]. A more accessible description of the rational reconstruction problem and the solution using Euclid's algorithm can be found in [8]. We use the algorithm of Monagan in [7] because it allows us to control the failure probability.

1.4.1 Maximal Quotient Rational Reconstruction

Theorem 1.4.1. [Wang, Guy, Davenport, 1982,[3]]. Let $n, d \in \mathbb{Z}$ with $d > 0$ and $\gcd(n, d) = 1$. Let $m \in \mathbb{Z}$ with $m > 0$ and $\gcd(m, d) = 1$. Let $u = n/d \bmod m$. Let $N, D \in \mathbb{Z}$ such that $N \geq n$ and $D \geq d$. Then

(i) if $m > 2ND$ the rational n/d satisfying the conditions above is unique, i.e., $\nexists a/b \in \mathbb{Q}$ also satisfying $\gcd(b, m) = 1, |a| \leq N, 0 \leq b \leq D$, $a/b \equiv u \bmod m$, and,

(ii) if $m > 2ND$ then on input of m and u there exists a unique index i in the Euclidean algorithm such that $r_i/t_i = n/d$. Moreover, i is the first index such that $r_i \leq N$.

Suppose we want to find a rational reconstruction of $u \pmod m$. By executing Euclidean algorithm on inputs $r_0 = m$ and $r_1 = u$, we obtain

$$\begin{aligned} r_0 &= q_2 r_1 + r_2 \\ r_1 &= q_3 r_2 + r_3 \\ \vdots &\quad \vdots \\ r_{l-1} &= q_{l+1} r_l + r_{l+1} \\ r_l &= q_{l+2} r_{l+1} + 0 \end{aligned}$$

where for each $2 \leq i \leq l+1$, $0 < r_i < r_{i-1}$, and q_i and r_i are the quotient and remainder of r_{i-2} divided by r_{i-1}.

Also, for any $2 \leq i \leq l+1$, the equation

$$r_i = t_i r_0 + s_i r_1$$

holds for some integers t_i, s_i, and the values of t_i and s_i can be obtained from the the extended Euclidean algorithm. Then for s_i and m are relatively prime, we obtain:

$$u \equiv r_i/s_i \pmod{m}.$$

Example 1.4.2. Suppose $n/d = 13/10$ and suppose we have computed n/d mod 997 and n/d mod 1009. Then we apply the Chinese remainder algorithm we obtain $u = 905377$ which satisfies $u \equiv n/d \bmod m$ where $m = 997 \times 1009 = 1005973$. If we apply Euclidean algorithm on u and m, we obtain

$$\begin{aligned} 1005973 &= 1 \times 905377 + 100596 \\ 905377 &= 9 \times 100596 + 13 \\ 100596 &= 7738 \times 13 + 2 \\ 13 &= 6 \times 2 + 1 \\ 2 &= 2 \times 1 + 0. \end{aligned}$$

We obtain these equations and rationals u' with $u' \equiv u \pmod{m}$:

i	q_{i+1}	s_i	r_i	$u' = r_i/s_i$
1	1	1	905377	905377
2	9	-1	100596	-100596
3	7738	10	13	13/10
4	6	-77381	2	-2/77381
5	2	464296	1	1/464296

The maximal quotient rational reconstruction algorithm outputs the rational r_i/s_i for which q_{i+1} is the maximal quotient, i.e., 13/10 in our example. The idea of this algorithm is to output the smallest rational r_i/s_i. Lemma 1.4.3 shows how the size of the quotient q_{i+1} relates to the size of the rational r_i/t_i and the modulus m over iterations of Euclidean algorithm.

Lemma 1.4.3 (Monagan,2004,[7]). Let $r_0 = m$ be the modulus and $r_1 = u$ be the image of a rational reconstruction, $\gcd(m, u) = 1$. By executing the Euclidean algorithm the inequality $m/3 < q_{i+1}|s_i|r_i \leq m$ holds for $2 \leq i \leq l + 1$ where q_{i+1} is the quotient in equation $r_{i-1} = q_{i+1}r_i + r_{i+1}$ and s_i is such satisfies $r_i = t_i r_0 + s_i r_1$.

The following lemma tells us that the algorithm is correct and there can only be one maximal quotient if m is large enough.

Lemma 1.4.4 (Monagan,2004,[7]). Let n,$d \in \mathbb{Z}$ with $d > 0$ and $\gcd(n, d) = 1$. Let $m \in \mathbb{Z}$ and $\gcd(m, d) = 1$. Let $u = n/d \bmod m$ and let i be an index with q_{i+1} a maximal quotient in the Euclidean algorithm when given input (m, u). Thus $u \equiv r_i/s_i \bmod m$. If $|n|d < \sqrt{m}/3$ then i is unique and $r_i/s_i = n/d$.

Now we know that the cost of rational number reconstruction is mainly the cost of Euclidean algorithm which is known to be $O(N^2)$, where $N = \log m$. Therefore, we try to reduce its complexity by recovering n/d using a small modulus m. It is easy to see that the smallest modulus m required to recover n/d is $m = 2|n|d$. Wang's algorithm [3] recovers n and d for $m > 2\max(|n|, d)^2$. The maximal quotient rational reconstruction algorithm (Monagan,2004,[7]) outputs n/d with high probability when the length of the modulus m is only a modest number of bits longer than the bits of nd. That is if $|n| \gg d$ or $d \gg |n|$ then the modulus needed by Wang's algorithm can be up to twice as long as that is needed by maximal quotient rational reconstruction algorithm.

We present here the maximal quotient rational reconstruction algorithm (MQRR) which takes inputs m, u, and T where T is the parameter that gives user control over the probability that the algorithm will succeed. This algorithm succeeds only if $q_{max} > T$.

1.4.2 Runtime Complexity of Rational Number Reconstruction

Both Wang's rational number reconstruction and the maximal quotient rational number reconstruction recover fractions by performing Euclidean algorithm. The cost of Euclidean algorithm is $O(N^2)$ where $N = \log_2 m$. In section 1.4.1, we have seen that the maximum quotient rational number reconstruction recovers n/d from input m and u for m slightly longer than $2|n|d$. Therefore, it costs $O(\log^2(nd))$ to successfully reconstruct the rational number n/d.

Algorithm: MQRR

Input: Integers $m > u \geq 0$ and $T > 0$.

Output: Either $n, d \in \mathbb{Z}$ s.t. $d > 0$, $\gcd(n, d) = 1$, $n/d \equiv u \pmod{m}$, and $T|n|d < m$, or FAIL.

1: If $u = 0$ then if $m > T$ then output 0 else output FAIL.
2: Set $(n, d) = (0, 0)$.
3: Set $(t_0, r_0) = (0, m)$.
4: Set $(t_1, r_1) = (1, u)$.
5: **while** $r_1 \neq 0$ and $r_0 > T$ **do**
6: Set $q = \lfloor 0/r1 \rfloor$.
7: If $q > T$ then set $(n, d, T) = (r_1, t_1, q)$.
8: Set $(r_0, r_1) = (r_1, r_0 - qr_1)$.
9: Set $(t_0, t_1) = (t_1, t_0 - qt_1)$.
10: **end while**
11: If $d = 0$ or $\gcd(n, d) \neq 1$ then output FAIL.
12: If $d < 0$ then set $(n, d) = (-n, -d)$.
13: Output (n, d).

1.5 A p-adic Lifting Algorithm to Solve $Ax = b$ over $\mathbb{Q}$

We will show how to solve $Ax = b$ over $\mathbb{Q}$ using p-adic lifting and rational reconstruction. The p-adic approach was first applied to linear systems by Dixon in [4] and Moenck and Carter in [5]. The recent paper of Chen and Storjohann [2] describes an implementation of this approach which reduces the matrix inversion modulo p to floating point matrix multiplications so that level 3 BLAS can be used. We first solve $Ax = b \bmod p$ for $x \in \mathbb{Z}_p^n$ then use p-adic lifting to obtain the solution $x \in \mathbb{Z}_{p^k}^n$ of $Ax \equiv b \pmod{p^k}$. Finally, we apply rational reconstruction to the entries of $x \bmod p^k$. The p-adic lifting algorithm was first tried by Hensel. The idea is based on the p-adic representation of the integers.

1.5.1 p-adic Representation of Integers

For any integer $u \in \mathbb{Z}$, we may write a unique representation of u such that

$$u = u_0 + u_1 p + u_2 p^2 + u_3 p^3 + \ldots + u_n p^n$$

where $p \geq 2$ is a positive integer, n is such that $p^{n+1} > 2|u|$, and $-\frac{p}{2} \leq u_i \leq \frac{p}{2}$ $(0 \leq i \leq n)$.

p-adic Representation of Integer Vectors

For an integer vector $V \in \mathbb{Z}^n$, we may also write V in the form

$$V = V_0 + V_1 p + V_2 p^2 + V_3 p^3 + \ldots + V_n p^n.$$

For example, Let $V = [9, -80, 94]^T$, and $p = 13$. We may obtain $V_0 = [-4, -2, 3]^T$, by the operation $V \pmod p$ in symmetric range, and $V_1 = \frac{V - V_0}{p} \pmod p = [1, -6, -6]^T$, and $V_2 = \frac{V - V_0 - V_1 p}{p^2} \pmod p = [0, 0, 1]^T$. Therefore, we obtain the unique p-adic representation $V = V_0 + V_1 p + V_2 p^2 = [-4, -2, 3]^T + 13[1, -6, -6]^T + 13^2[0, 0, 1]^T$.

Solving Linear Systems of Equations $Ax = b$ over $\mathbb{Q}$

We now apply p-adic lifting algorithm to solve linear systems of equations over $\mathbb{Q}$. For example, let $A = \begin{bmatrix} -25 & -44 & 86 \\ 51 & 24 & 20 \\ 76 & 65 & -61 \end{bmatrix}$, $b = [1, 2, 3]^T$, and $p = 13$. We would like to find vector x such that $Ax = b$.

First of all, let us show a general solution to this system. Let $x^{(k)} = x_0 + x_1 p + \cdots + x_{k-1} p^{k-1}$ be the solution of $Ax \equiv b \bmod p^k$, i.e., $x^{(k)}$ is the k^{th} order approximation of x. Therefore, we can find the first order approximation $x^{(1)} = x_0$ by solving the equation $Ax_0 \equiv b \bmod p$. Assuming we know $x^{(k)}$ for $k \geq 1$, the $(k+1)^{th}$ order approximation, i.e., $x^{(k+1)}$, can be determined by the k^{th} order approximation $x^{(k)}$ and the equation $Ax^{(k+1)} = b \bmod p^{k+1}$ from

$$Ax^{(k+1)} = A(x^{(k)} + x_k p^k) \equiv b \bmod p^{k+1}.$$

Since A, b, and $x^{(k)}$ are known, we can then obtain x_k from the above equation hence $x^{(k+1)}$. In our example, the first order approximation $x^{(1)} = x_0 = [4, 5, 6]^T$ is obtained by solving the equation $Ax_0 = b \pmod p$ in the symmetric range. Then we obtain $x_1 = [-5, 3, -1]$ from the equation $Ax_1 = \frac{b - Ax^{(1)}}{p} \pmod p$, and so on. Noting that, the solution vector is lifted to $\mathbb{Z}_{p^k}^n$ in the k^{th} iteration, however, we may never balance the equation $A(x_0 + x_1 p + \ldots + x_l p^l) = b$ for any $l \in \mathbb{Z}$ if the true solution vector x has fractions in its entries. Rational number reconstruction is used to solve this problem. We apply rational number reconstructions to $x^{(k)} \bmod p^k$ for $k = 1, 2, 3, \ldots$ to obtain $y^{(k)} \in \mathbb{Q}^n$. We stop when $Ay^{(k)} = b$. In our example, we try to compute images of x and do rational reconstruction till

$x^{(7)} = x_0 + x_1 p + x_2 p^2 + x_3 p^3 + x_4 p^4 + x_5 p^5 + x_6 p^6$ which we successfully recovered the fraction entries using maximal quotient rational reconstruction and obtain $x = [\frac{964}{1073}, -\frac{3089}{2146}, -\frac{995}{2146}]^T$.

1.6 Other Definitions, Results and Notations Used

1.6.1 Definitions and Notations

We now define some notations and state some techniques that will be used in the implementation and analysis of our modular algorithms.

Let $m(z)$ be a polynomial in z with integer coefficients. We denote the maximum of the absolute value of coefficients in $m(z)$ by $||m||_\infty$.

Definition 1.6.1. We define in our paper that the length (or size) of a rational number n/d is the length of the absolute value of the product of n and d, i.e., $\log |nd|$.

Let M be a matrix or vector of polynomials with rational coefficients. Let n/d be the rational coefficient such that is the largest in length. We denote the value $|nd|$ by $||M||_\infty$.

Single-Point Evaluation and Interpolation

Consider the problem of computing $c(z) = a(z) \times b(z) \in \mathbb{Z}[z]$, where $a(z) = z^3 + 5z^2 + 3z + 6$, $b(z) = 2z^2 + 3z + 1$. Let $\zeta \in \mathbb{Z}$ be a positive integer which bounds $2||c(z)||_\infty$. We substitute ζ into $a(z)$ and $b(z)$ and compute their product. We choose $\zeta = 1000$ for simplicity. We have, $a(\zeta) = 1005003006$, $b(\zeta) = 2003001$, hence $c(\zeta) = a(\zeta) \times b(\zeta) = 1005003006 \times 2003001 = 2013022026021006$. Now, we do single-point interpolation at $z = 1000$ and obtain $c(z) = 2z^5 + 13z^4 + 22z^3 + 26z^2 + 21z + 6$ from $c(\zeta)$. Is $c(z)$ precisely the product of $a(z)$ and $b(z)$? It must be if $\zeta > 2||c||_\infty$.

Remark: One should choose $\zeta = B^m$ where B is the base of the integer system so that evaluation and single-point interpolation are linear time. This method works the same for polynomials with negative coefficients if we use symmetric range in the interpolation step.

Chapter 2

Algorithms

2.1 Gaussian Elimination Approach

2.1.1 Description

As we have discussed in Chapter 1, Gaussian elimination may be used to solve linear systems over the rationals. In this section, we will use Gaussian elimination to solve linear systems of equations over cyclotomic fields and discuss its runtime complexity. Let $m(z) = \Phi_k(z)$ be the cyclotomic polynomial of order k. Let $d = \deg m(z)$ and let $F = \mathbb{Q}[z]/m(z)$ be a cyclotomic field. Since the minimal polynomial $m(z)$ is irreducible over $\mathbb{Q}[z]$, for $a(z) \in F \backslash \{0\}$, we can always find a unique inverse $a^{-1}(z) \in F$ by applying the extended Euclidean algorithm. Let $A \in F^{n \times n}$, $b \in F^n$ be the input matrix and vector. We are able to perform row reductions to reduce the system, hence obtain the solution vector $x \in F^n$ which satisfies the equation $Ax = b$. We assume the inputs A and b have integer coefficients, and the entries of A and b have been reduced by $m(z)$. This is easy to achieve by multiplying each equation the least common multiple of the rational coefficients in the equation. Therefore, our inputs satisfy $A \in \mathbb{Z}[z]^{n \times n}/m(z)$, and $b \in \mathbb{Z}[z]^n/m(z)$.

This straight forward approach is simple, easy to code and ideal for very small systems, e.g., small matrix dimensions and low degree minimal polynomials.

Example 2.1.1. Let $A = [33z + 2]$, $b = [22z - 55]$ and $m(z) = \Phi_4[z] = z^2 + 1$. First, we compute the inverse of $33z + 2$ which is $-\frac{33}{1093}z + \frac{2}{1093}$. Then, we compute

$$x = [(22z-55)(-\frac{33}{1093}z+\frac{2}{1093})] = [-\frac{726}{1093}z^2+\frac{1859}{1093}-\frac{110}{1093}] \equiv [\frac{1859}{1093}z+\frac{616}{1093}] \pmod{z^2+1}.$$

In calculating x, we run the Euclidean algorithm in the ring $\mathbb{Q}[z]$, then do polynomial multiplication, and finally a polynomial division by $m(z)$. We can see in our example that the maximum coefficient length in the input matrix A and vector b is 2 decimal digits. However, during our calculation, the maximal coefficient length in A^{-1} increases to 6 decimal digits, and the maximum coefficient length in the solution vector increases to 7 decimal digits. For large random inputs, i.e., n large, d large, we may get coefficients in solution vectors with nd times the length of the maximal coefficient length in the inputs.

2.1.2 Runtime Analysis

Before we analyze the runtime, we define some notations for our problem. Let $A \in \mathbb{Z}[z]^{n\times n}/m(z)$ and $b \in \mathbb{Z}[z]^n/m(z)$ be our input matrix and vector, and $m(z) = \Phi_k(z)$ be the minimal polynomial. Let $c = \log\max(||A||_\infty, ||b||_\infty)$ denote the maximum coefficient length in the input matrix and vector and let $d = \deg m(z)$. We assume that the maximum length of the rational coefficients in the solution vector $x \in \mathbb{Q}[z]^n/m(z)$ is L, that is $\log||x||_\infty \in O(L)$.

Since we know that Gaussian elimination over $\mathbb{Q}$ involves $O(n^3)$ multiplications over $\mathbb{Q}$, we expect $O(n^3)$ multiplications in the field $F = \mathbb{Q}[z]/m(z)$ and $O(n)$ calls to the Euclidean algorithm for inverses in F. Assuming classical algorithms for polynomial arithmetic, Gaussian elimination over F costs $O(n^3d^2)$ arithmetic operations over $\mathbb{Q}$, but the size of the rationals grows. Each polynomial multiplication takes $O(d^2l^2)$ operations where l is the maximum coefficient length of the polynomials. We take $l = \frac{L}{2}$ which is the average length of the polynomial coefficients in the computation, and obtain its runtime complexity to be $O(n^3d^2L^2)$ using classical, i.e., quadratic, algorithms for integers and polynomials.

2.1.3 Reduction to Solving over $\mathbb{Q}$

The solution vector $x \in F^n$ of the linear system in section 2.1.1 is a vector of polynomials in z of degree $\leq d-1$ over $\mathbb{Q}$ where $d = \deg m(z)$. Writing $x_i = \sum_{j=0}^{d-1} a_{ij}z^j$ for $i = 1, 2, \ldots, n$ for unknown coefficients a_{ij}, if we multiply out $Ax - b = 0$ and divide by $m(z)$ and equate coefficients of z^j to zero, we obtain an nd by nd linear system over $\mathbb{Q}$. This can be solved using Gaussian elimination in $O(n^3d^3)$ arithmetic operations over $\mathbb{Q}$. This reduction to $\mathbb{Q}$ is not a good method. It increases the cost by a factor of d arithmetic operations over $\mathbb{Q}$ when compared with the direct method in section 2.1.1.

2.2 Chinese Remaindering Approach

We have stated the Chinese remainder theorem and we showed an algorithm to solve Chinese remaindering problem in section 1.3. We have also shown an example of using Chinese remaindering to solve linear systems over the integers. In this section, we will discuss how to solve linear systems of equations over cyclotomic fields by using Chinese remaindering and rational number reconstruction.

2.2.1 The Subroutines

Finding Primes

Let $m(z) = \Phi_k(z)$ denote our minimal polynomial. Thus $m(z)$ is the k^{th} cyclotomic polynomial which is irreducible over $\mathbb{Q}$. The fundamental theorem of arithmetic says that every integer can be uniquely factored into a product of primes. Similarly, in the field $F = \mathbb{Z}_p$, where p is a prime, $m(z)$ can be factored into a product of irreducible polynomials over F which have degree less than or equal to the degree of $m(z)$. In our problem, we would like to find primes p such that $m(z)$ factors into distinct linear factors over $\mathbb{Z}_p$. For example, $m(z) = \Phi_5(z) = z^4 + z^3 + z^2 + z + 1 = (z+2)(z+6)(z+7)(z+8) \bmod 11$. If we do this then we can solve $Ax = b \bmod p$ at each root of $m(z)$ independently.

Lemma 2.2.1. Let $m(z)$ be the k^{th} cyclotomic polynomial and p be a prime such that $p \nmid k$. We have that $p \equiv 1 \pmod{k}$ if and only if $m(z)$ has roots in $\mathbb{Z}_p$.

Proof. Recall that if p is a prime, then Fermat's little theorem implies $a^p \equiv a \bmod p$ for all integers a. Hence, $0, 1, 2, ..., p-1$ are roots of the polynomial $z^p - z$ over $\mathbb{Z}_p$. Since $m(z)|z^k - 1$, to prove the Lemma it suffices to show $z^k - 1|z^{p-1} - 1$ over $\mathbb{Z}_p$ if and only if $k|p-1$. The easiest way to see this is to verify that if $p - 1 = kq$ then

$$z^{p-1} - 1 = z^{kq} - 1 = (z^k - 1)(z^{k(q-1)} + z^{k(q-2)} + ... + z^k + 1)$$

and if $p-1 = kq+r$ with remainder $r \neq 0$ then the remainder of $z^{kq+r} - 1$ divided by $z^k - 1$ is $z^r - 1$ which is not zero over $\mathbb{Z}_p$. □

The following lemma can be derived from lemma 2.2.1 and Direchlet's theorem (see Chapter 9 of [1]).

Lemma 2.2.2. Let $m(z)$ be a cyclotomic polynomial with degree d. Primes p_i's which split $m(z)$ into distinct linear factors over $\mathbb{Z}_{p_i}$ exist frequently. The probability that such primes exist is about 1 in every d primes. In general, $\sim \frac{1}{d!}$ primes split an irreducible polynomial of degree d.

Lemma 2.2.2 tells us in theory how frequently the primes split $m(z)$ into distinct linear factors. Lemma 2.2.1 tells us a method for how to compute the next suitable prime fast if we obtain one such prime. Therefore, we need an algorithm to find the first prime such that splits $m(z)$ into distinct linear factors. We simply start with the biggest machine prime that we can use then try its previous primes one by one until we get one that satisfies lemma 2.2.1.

Spliting the Minimal Polynomial

Let $m(z) = \Phi_n(z)$ and let $p \equiv 1 \pmod{n}$ be primes. Lemma 2.2.1 says $m(z)$ splits mod p. To split $m(z)$ mod p, we use the following method.

Fermat's little theorem implies that the polynomial $p_r(z) = z^p - z = (z-0)(z-1)\cdots(z-(p-1))$ in $\mathbb{Z}_p$. We use the probabilistic algorithm of Rabin in [9] which is based on the following idea. For a prime $p > 2$, the polynomial $z^p - z = z(z^{(p-1)/2} - 1)(z^{(p-1)/2} + 1)$ has roots $0, 1, 2, \ldots, p-1$ in $\mathbb{Z}_p$. Therefore, for any $\alpha \in \mathbb{Z}$, the polynomial $(z+\alpha)^{(p-1)/2} - 1$ has $(p-1)/2$ of the integers in $\{0, 1, 2, \ldots, p-1\}$ as the roots in $\mathbb{Z}_p$. Hence, if we compute $g = \gcd((z+\alpha)^{\frac{p-1}{2}-1} - 1, m(z))$, we will likely to get a non-trivial factor of $m(z)$ with probability over $1/2$. By repeating this GCD computation for randomly chosen $\alpha \in \mathbb{Z}_p$, we will eventually split $m(z)$ (see [11], Chapter 8). Then we have $m = g \times m/g$. We recursively split g and m/g until we find all the roots.

We use the runtime analysis of Rabin's algorithm from Gerhard and von zur Gathen's book [10] in Theorem 14.9 and have the following result.

Theorem 2.2.3. Let $m(z) = \Phi_n(z)$ and $d = \deg m(z) = \varphi(n)$. The expected number of arithmetic operations in $\mathbb{Z}_p$ that Rabin's algorithm takes to split $m(z)$ into linear factors over $\mathbb{Z}_p$ is $O(\log d(\log p + \log d)M(d))$ where $M(d)$ is the cost of multiplying two polynomials of degree d over $\mathbb{Z}_p$.

Note, the first contribution to the cost, $\log d \log p\, M(d)$, is the repeated squaring cost and the second contribution, $\log^2 d\, M(d)$, is the GCD computation cost. If one uses classical quadratic algorithms for univariate polynomial multiplication and GCD, the expected running time is $O(\log p\, d^2 \log d)$.

Rational Number Reconstruction

In Chapter 1 we have discussed how to recover a rational number with fractions from its image. We apply this technique to recover the rational coefficients of a polynomial from its image modulo m. For example, let $p_1 = 15412z^3 + 21025z^2 + 7713z + 13504, m_1 = 23117$ be an image polynomial. We would like to find polynomial $p \in \mathbb{Q}[z]$ such that $p \equiv p_1 \pmod{m_1}$. By executing maximal quotient rational reconstruction on each coefficient independently, we get $\frac{2}{3} \equiv 15412 \pmod{23117}$, $\frac{105}{11} \equiv 21025 \pmod{23117}$, $\frac{22}{3} \equiv 7713 \pmod{23117}$, and $\frac{1}{101} \equiv 13504 \pmod{23117}$ to be the coefficients of our original polynomial $p = \frac{2}{3}x^3 + \frac{105}{11}x^2 + \frac{22}{3}x + \frac{1}{101}$.

2.2.2 The Algorithm

By utilizing the above algorithms along with the polynomial evaluation and interpolation algorithms that we have discussed in section 1.2, we can now present our first main algorithm. Figure 2.1 shows the main process flow of the Chinese remaindering approach. We divide the process into 5 main phases. The first phase is to choose primes such that our minimal polynomial $m(z)$ can be factored into distinct linear factors, and then reduce the coefficients in the input A, b modulo those primes. The second phase is to compute all the roots of $m(z)$ with respect to the primes that we chose and evaluate the input matrix and vector by polynomial evaluation. The third phase is to solve the modulo integer systems over $\mathbb{Z}_{p_i}$ using Gaussian elimination. The fourth phase is to do the polynomial interpolation over z, our variable, to obtain our image polynomials. The fifth phase, which is the final calculation, is to recover the image polynomials over all primes using Chines remaindering algorithm and then perform rational number reconstruction to recover the rational coefficients in the solution vector. We stop when the result y produced by rational reconstruction satisfies $Ay = b$. Algorithm 1 shows the detailed algorithm of this approach.

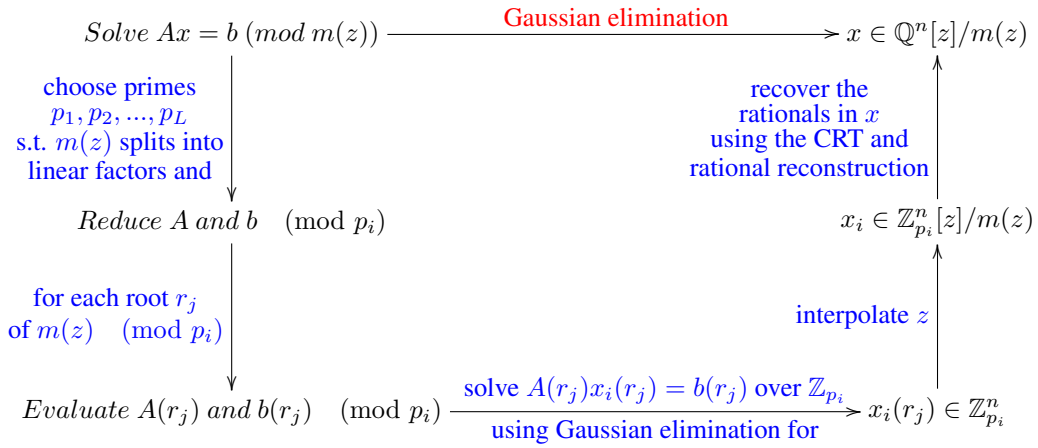

Figure 2.1: Process flow of the Chinese remaindering approach

Algorithm 1 Algorithm for the **CRT Approach**

Input: $A \in \mathbb{Z}[z]^{n \times n}/m(z)$, $b \in \mathbb{Z}[z]^n/m(z)$, $m(z) \in \mathbb{Z}[z]$, $\det(A) \not\equiv 0 \pmod{m(z)}$
Output: $x \in \mathbb{Q}[z]^n$ which satisfies $Ax = b \pmod{m(z)}$

1: Let $X = 0$, $P = 1$.
2: **for** $k = 1, 2, 3, \ldots$ **do**
3: Find a new machine prime p_k, s.t. $m(z)$ splits linearly over $\mathbb{Z}_{p_k}$, and compute the roots $\alpha_1, .., \alpha_d$ of $m(z) \bmod p_k$.
4: Let $A_k = A \bmod p_k$ and $b_k = b \bmod p_k$.
5: **for** $i = 1$ to d **do**
6: Substitute α_i into A_k and b_k.
7: Solve the linear system $A_k(\alpha_i)x_{ki} = b_k(\alpha_i) \bmod p_k$ for x_{ki}.
8: **if** $A_k^{-1}(\alpha_i)$ does not exist **then** Goto step 3 **end if**
9: **end for**
10: Interpolate $x_k \in \mathbb{Z}_{p_k}[z]^n$ using $x_{k1}, .., x_{kd}$ wrt. $\alpha_1, .., \alpha_d$.
11: Set $X = CRT([X, x_k], [P, p_k])$, $P = P \times p_k$.
12: **if** $k \in \{1, 2, 4, 8, 16, \ldots\}$ **then** Set $x = RR(X, P)$ **end if**
13: **if** $x \neq FAIL$ **and** $m(z) | Ax - b$ **then** Output x **end if**
14: **end for**

2.2.3 Correctness of Algorithm 1

As we have mentioned in section 2.1, we assume in our algorithm that the input matrix and vector will have polynomial entries reduced by the minimal polynomial and with integer coefficients. We also assume that A is invertible over $\mathbb{Q}[z]/m(z)$. In order to prove that Algorithm 1 is correct, we need to show that all images of the solutions used in the reconstruction of the solution x over $\mathbb{Q}[z]$ are correct. Consider the 1 by 1 linear system

$$[10z + 15]x = [1]$$

where $m(z) = z^2 + z + 1$. The solution is

$$x = [-2/35z + 1/35].$$

Looking at the solution we see that our algorithm cannot work if we choose primes 5 or 7. It is clear that the matrix $A = [10z + 15]$ is singular modulo 5 and Algorithm 1 detects this in step 8. But what about the prime 7? The determinant $D = \det A = 10z + 15$ is not 0 modulo 7 but D^{-1} does not exist modulo 7 and hence A is not invertible modulo 7. Does Algorithm 1 also eliminate the prime 7? Lemma 2.2.5 below shows that Algorithm 1 does eliminate the prime 7 in the above case. First a definition.

Definition 2.2.4. Let $D = \det(A) \in \mathbb{Z}[z]$. A prime p chosen by Algorithm 1 is said to be *unlucky* if D is invertible modulo $m(z)$ but D is not invertible modulo $m(z)$ modulo p.

Lemma 2.2.5. Let p be a prime chosen in Algorithm 1 so that $m(z) = \Pi_{i=1}^{d}(z - \alpha_i)$ for distinct $\alpha_i \in \mathbb{Z}_p$. Then p is unlucky $\Rightarrow A(\alpha_i)$ is not invertible modulo p for some i.

Proof. Let $D = \det A \in \mathbb{Z}[z]$. Then p is unlucky $\Rightarrow D$ is not invertible modulo $(m(z), p) \Rightarrow \deg_z \gcd(D \bmod p, m \bmod p) > 0 \Rightarrow (z - \alpha_i) | D \bmod p$ for some $i \Rightarrow D(\alpha_i) = 0 \bmod p \Rightarrow A(\alpha_i)$ is not invertible mod p (for some i). □

From the proof we can see also that the unlucky primes are precisely the primes that divide the resultant

$$R = \mathrm{res}_z(D(z), m(z)).$$

It follows that for given inputs A, b and $m(z)$ with A invertible in characteristic 0, there are finitely many unlucky primes, and therefore, if the primes chosen by Algorithm 1 are chosen from a sufficiently large set, Algorithm 1 will rarely encounter an unlucky prime. The proof

of Theorem 2.4.2 in section 2.4 bounds the size of the integer R and can be used to bound the probability that Algorithm 1 chooses an unlucky prime. It can also be used to modify Algorithm 1 to detect whether A is singular in characteristic 0. If A is singular, it follows that $A_k^{-1}(\alpha_i)$, in step 8, does not exist for all primes. Let $P = \prod p_i$ be the product of primes that has been determined "unlucky" in step 8 of Algorithm 1, we can conclude that A is singular in characteristic 0 if $P > ||\det A \bmod m(z)||_\infty$.

In our analysis of the running time of Algorithm 1 below we have assumed that unlucky primes are rare, and hence, do not affect the running time. Our implementation of Algorithm 1 uses machine primes, 31 bit primes on a 64 bit machine, and 25 bit floating point primes on a 32 bit machine, and consequently unlucky primes are rare in practice.

In step 11, we show the method of updating X by performing incremental Chinese remaindering in each iteration. However, the complexity of obtaining X can be reduced by doing partially recursive Chinese remaindering (see Proposition 2.2.13) since we perform rational reconstruction on X when k is a power of 2, i.e., $k \in \{1, 2, 4, 8, 16, \ldots\}$.

2.2.4 Runtime Analysis of Algorithm 1

As mentioned previously, we assume $A_{ij}, b_i \in \mathbb{Z}[z]/m(z)$ with degree $< d = \deg m(z)$, and let $c = \log \max(||A||_\infty, ||b||_\infty)$ be the maximum length of the integer coefficients in the inputs, and $d = \deg m(z)$ be the degree of our minimal polynomial. In addition, we assume that we need L machine primes to successfully recover the rational coefficients in the solution vector. In our later analysis, we state the running time of Algorithm 1 in terms of just the variables n, d, and c. Because we use machine primes, i.e., primes of constant bit-length that fit into a machine word, we know that L is linear in $\log ||x||_\infty$, the length of the largest rational coefficient in x.

In general, the length of the rationals appearing in the output can be slightly more than nd times longer than those in the input (see Theorem 2.4.2). However, our linear systems arising in practice (Table 2.2) show that L can be much much smaller. Thus we state the running times for L and also for $L \in O(ndc + nd^2)$ in section 2.5.

Proposition 2.2.6. Reducing the coefficients of the input matrix A and vector b modulo our chosen prime p takes $O(n^2dc)$ operations.

Proof. There are n^2+n polynomials with degree $d-1$ to reduce. Therefore, we have n^2d+nd coefficients to reduce modulo p independently. Each reduction is a modulo operation on an integer coefficient with maximum possible length c and a prime p with length $O(1)$, i.e., constant length. Therefore, the cost in total is $O(n^2dc)$. □

Proposition 2.2.7. Applying polynomial evaluation to substitute d roots into A and b modulo p takes $O(n^2d^2)$ word operations.

Proof. The coefficients in input matrix A and vector b we are considering here have been reduced modulo p before doing the polynomial evaluation. Therefore, there are $n^2 + n$ polynomials of degree $< d$ with coefficients in $\mathbb{Z}_p$ to be evaluated. We evaluate using Horner form each polynomial with maximum degree $d - 1$ at d points $\alpha_i \in \mathbb{Z}_p$. Each polynomial evaluation costs $O(d)$ operations in $\mathbb{Z}_p$. Therefore, the total cost becomes $O(n^2d^2)$ word operations. □

Proposition 2.2.8. Solving the linear system $A(\alpha)x(\alpha) = b(\alpha) \bmod p$, where α is one of the roots of $m(z)$ in $\mathbb{Z}_p$, for $x(\alpha)$ over all d roots of $m(z) \pmod p$ takes $O(n^3d)$ word operations.

Proof. Solving each linear system $A(\alpha)x(\alpha) = b(\alpha) \bmod p$ takes $O(n^3)$ operations in $\mathbb{Z}_p$ by applying Gaussian elimination. There are d such systems to solve over $\mathbb{Z}_p$. Therefore, the total cost becomes $O(n^3d)$ word operations. □

Proposition 2.2.9. Doing polynomial interpolation to construct $x \in \mathbb{Z}_p[z]^n$ from a series of $(\alpha_i, x(\alpha_i)) \in (\mathbb{Z}_p, \mathbb{Z}_p^n)$ takes $O(nd^2)$ word operations.

Proof. We have discussed polynomial interpolation over F in section 1.2. It costs $O(d^2)$ operations in F to interpolate a polynomial with degree $d - 1$ from d evaluation points. Therefore, it costs $O(nd^2)$ operations over $\mathbb{Z}_p$ to interpolate $x \in \mathbb{Z}_p[z]^n$ if each polynomial interpolation is done independently from others. □

Runtime Complexity of Doing Chinese Remaindering

Unlike the runtime complexities of the procedures in phase one, two, three, and four, the cost of the Chinese remaindering procedure varies in each iteration regardless the fact that we use primes of the same length. We discuss two ways to implement the Chinese remaindering

in our problem. One is incremental Chinese remaindering, and the other is recursive Chinese remaindering.

Proposition 2.2.10. Let p_1 be an integer of constant bit length, e.g., a machine prime, and p_2 be an integer of length n, e.g., p_2 has n times the length of a machine prime, where $\gcd(p_1, p_2) = 1$. Let $a, b \in \mathbb{Z}$ satisfy $0 \leq a \leq p_1$, $0 \leq b \leq p_2$. It takes $O(n)$ operations to apply Chinese remainder algorithm to compute the integer $c \in \mathbb{Z}$ satisfying $c \equiv a \pmod{p_1}$, $c \equiv b \pmod{p_2}$, and $0 \leq c \leq p_1 p_2$. The result c is an integer of length at most $n + 1$, i.e., c has length at most $n + 1$ times the length of a machine prime.

Proof. This is a problem of doing Chinese remaindering on two images. $a \in \mathbb{Z}_{p_1}$ is the image which has constant bit length, and $b \in \mathbb{Z}_{p_2}$ is the image which has $O(n)$ times the bit length of a. We want to find $c \equiv a \pmod{p_1}$, and $c \equiv b \pmod{p_2}$. We know from Theorem 1.3.1 that such integer c exists in $\mathbb{Z}_{p_1 \times p_2}$. We use mixed radix representation to solve this problem. Let $c = c_0 + c_1 p_2$. Reducing this equation by p_2, we get $c_0 = b$. Reducing the equation by p_1, we get $c_1 = (a - b)p_2^{-1} \bmod p_1$. Therefore, we found such $c \in \mathbb{Z}_{p_1 \times p_2}$. The cost of finding c is just the cost of finding $p_2^{-1} \pmod{p_1}$, multiplying $(a - b)$ by p_2^{-1} and compute $c_0 + c_1 p_2$. Finding the inverse of p_2 is done by reducing $p_2 \bmod p_1$, and then finding the inverse. Since p_1 has constant bit length, the cost of inversion is constant. Therefore, the total cost of this problem is dominated by an integer division between a length $O(n)$ integer and a length $O(1)$ integer, i.e., $p_2 \bmod p_1$, and integer multiplications between length $O(n)$ integers and length $O(1)$ integers which cost $O(n)$ word operations. The second part of this proposition follows directly from the fact that $0 \leq c \leq p_1 p_2$. □

Proposition 2.2.11. Let p_1 and p_2 be integers of length n, e.g., n times the length of a machine prime, and $\gcd(p_1, p_2) = 1$. It takes $O(n^2)$ operations to apply Chinese remaindering to compute the integer $c \in \mathbb{Z}$ satisfying $c \equiv a \pmod{p_1}$, $c \equiv b \pmod{p_2}$, and $0 \leq c \leq p_1 p_2$. The result c is an integer of length at most $2n$, i.e., c has length at most $2n$ times the length of a machine prime.

Proof. Similar to the proof of Proposition 2.2.10, we can use mixed radix representation to solve this problem. However, it costs more than $O(n)$ operations to compute the inverse of p_2 over $\mathbb{Z}_{p_1}$ in this case. We know that both p_1 and p_2 are integers with length n, and we need to run Euclidean algorithm to find $p_2^{-1} \bmod p_1$ which costs $O(n^2)$. The rest of calculation involve multiplications between length $O(n)$ integers, e.g., $(a - b) \times p_2$, which

also cost $O(n^2)$ operations assuming classical integer multiplication. Therefore, the cost of this problem is $O(n^2)$ and the second part of this proposition follows directly from the fact that $0 \leq c \leq p_1 p_2$. □

Proposition 2.2.12 (Cost of Incremental Chinese Remaindering)**.** The runtime complexity of doing incremental Chinese remaindering to construct $x \in \mathbb{Z}_{p_1 \times p_2 \times \ldots \times p_L}[z]^n$ from $x_1 \in \mathbb{Z}_{p_1}[z]^n$, $x_2 \in \mathbb{Z}_{p_2}[z]^n$,...,$x_L \in \mathbb{Z}_{p_L}[z]^n$ is $O(ndL^2)$.

Proof. We are doing incremental Chinese remaindering on L machine primes each with length 1, i.e., the length of a machine prime. We can learn from Theorem 2.2.10 that the calculation involving p_1 and p_2 costs $O(nd \cdot 1)$ operations and the resulting vector $x \in \mathbb{Z}_{p_1 \times p_2}[z]^n$ has coefficients of length 2, i.e., twice the length of a machine prime. The next calculation involving the previous primes and the new prime p_3 costs $O(nd \cdot 2)$ operations. Therefore, the cost of calculating $x \in \mathbb{Z}_{p_1 \times p_2 \times \ldots \times p_L}[z]^n$ is $O(nd(1 + 2 + 3 + \ldots + L - 1)) = O(ndL^2)$ word operations. □

Proposition 2.2.13 (Cost of Recursive Chinese Remaindering)**.** The runtime complexity of doing recursive Chinese remaindering to construct $x \in \mathbb{Z}_{p_1 \times p_2 \times \ldots \times p_L}[z]^n$ from $x_1 \in \mathbb{Z}_{p_1}[z]^n$, $x_2 \in \mathbb{Z}_{p_2}[z]^n$, ..., $x_L \in \mathbb{Z}_{p_L}[z]^n$ is $O(ndM(L) \log L + L^2)$, where $M(L)$ is the cost of fast integer multiplication.

Proof. In the recursive version of Chinese remaindering, we perform Chinese remainder algorithm on a pair of vectors u, v modulo a pair of integers P and Q. For example, at step $k = 2^{j+1}$, we are recovering integers modulo $P = p_1 \times p_2 \times \ldots \times p_{2^j}$, $Q = p_{2^j+1} \times p_{2^j+2} \times \ldots \times p_{2^{j+1}}$. This requires inverting P modulo Q which costs $O((2^j)^2)$ using the classical Euclidean algorithm. However, this is done just once for all pairs of integer coefficients in vectors v_1 and v_2. Then the rest is to do scalar multiplication of the vector $v_1 - v_2$ by $P^{-1} \bmod Q$ which costs $O(ndM(2^j))$ operations where $M(k)$ is the cost of multiplying and dividing integers of length k. It would have no gain in comparison to the incremental Chinese remaindering if there is only classical integer multiplication and division is available. If a fast integer multiplication algorithm, e.g., FFT, is available, this reduces the total cost of recursive Chinese remaindering from $O(ndL^2)$ to $O(ndM(L) \log L + L^2)$. □

Runtime Complexity of Rational Reconstruction

Unlike all of the above procedures, rational number reconstruction tries to construct the final answer and hence decide if we need more primes. Because we do not predict the number of primes that are needed to successfully construct all coefficients in the solution vector, we try rational reconstruction at certain points to test if we have used enough primes. As a result, we have two parts which should be included in the total runtime complexity of the rational reconstruction. We called them "unsuccessful rational reconstruction" for the intermediate trials which return "FAIL", and "successful rational reconstruction" which successfully returns the solution vector $x \in \mathbb{Q}[z]/m(z)$. The same assumption is made here that we will use L primes to successfully reconstruct the rational coefficients in the solution vector.

Proposition 2.2.14 (Cost of Unsuccessful Rational Reconstruction)**.** The runtime complexity of unsuccessful rational reconstruction is, in the worst case, $O(ndL^2)$ by attempting rational reconstruction after $j = 1, 2, 4, 8, ...$ primes.

Proof. There are at most $\log_2 L$ unsuccessful attempts of rational reconstruction if we try it after $j = 1, 2, 4, 8, ...$ primes. In the i^{th} iteration, we do rational reconstructions on image coefficients of the solution vector over $\mathbb{Z}_{p_1 \times p_2 \times \cdots \times p_i}$, which are integers approximately i times the length of a machine prime that is used. In the worst case, each attempt to reconstruct x fails at the very last coefficient. Therefore, the cost of unsuccessful rational reconstruction becomes

$$nd \sum_{i=0}^{\log_2 L - 1} O(2^{2i}) \in O(ndL^2).$$

Furthermore, in the best case, the length of the coefficients in the solution vector are similar, and rational reconstruction is unsuccessful on the first coefficient. Therefore, the complexity of the unsuccessful attempts will be bounded by $O(L^2)$. □

Proposition 2.2.15 (Cost of Successful Rational Reconstruction)**.** The cost of successful rational number reconstruction in algorithm 1 is dominated by the cost of rational number reconstruction in the last iteration which produces our true solution vector $x \in \mathbb{Q}[z]^n/m(z)$ from L primes such that $Ax \equiv b \pmod{m(z)}$. Furthermore, it has runtime complexity $O(ndL^2)$.

Proof. Whenever the rational number reconstruction procedure produces a result x other than "FAIL", it has successfully reconstructed all nd coefficients in the solution vector $x \in \mathbb{Q}[z]^n/m(z)$. This vector x may or may not be our true solution vector which satisfies the equation $Ax = b \pmod{m(z)}$. The complexity of rational reconstruction is $O(N^2)$, (see section 1.4) where N is the length of the rational modulus m. Therefore, the cost of rational reconstruction in the last iteration which produces our true solution vector x is $O(ndL^2)$ because each rational number reconstruction has to be done independently and the modulus is the product of L machine primes. Since we perform the rational reconstruction in only the k^{th} iterations, where $k = 2^i, i \in \mathbb{Z}^+ \cup \{0\}$, the cost of successful rational reconstructions is $\sum_{i=0}^{\log_2 L} ndO(2^{2i}) \in O(ndL^2)$ if we assume every attempt of rational number reconstruction returns a vector x rather than "FAIL". Hence, the result follows. □

Remark: The cost of the successful rational reconstruction of the $\leq nd$ rational coefficients in x can similarly be reduced to roughly one rational reconstruction and $O(nd)$ long multiplications and divisions using a clever trick. Suppose we are reconstructing a rational from an image $u \bmod P$ and b is the LCM of the denominators of all rationals reconstructed so far. The idea is to apply rational reconstruction to $b \times u \bmod P$ instead (see [2] for details). Assuming fast integer multiplication and division are available, this improvement effectively reduces the cost of rational reconstruction to that of fast multiplication, that is, from $O(ndL^2)$ to $O(L^2 + ndM(L))$ where $M(L)$ is the cost of multiplication of integers of length L and the L^2 term is the cost of the classical Euclidean algorithm which we use for computing inverses and rational reconstruction.

Proposition 2.2.16. Checking if $m(z)|Ax - b$ takes $O(n^2d^2cL)$ operations.

Proof. This checking involves a matrix vector multiplication of polynomials with coefficient lengths c and L, which costs $O(n^2d^2cL)$. This cost dominates the cost of other operations in this checking procedure. □

Theorem 2.2.17. The running time complexity of the Chinese remaindering approach is $O(n^3dL + n^2d^2cL + ndL^2)$.

Remark: The complexity of the checking procedure seems to dominate the cost of all procedures in Algorithm 1 except the Gaussian elimination, Chinese remaindering, and rational reconstruction. However, in practice, we improve its efficiency by clearing all fractions in x

before the matrix vector multiplication, and observed its cost never exceed 10% of the total running time.

2.2.5 Run Out of Primes Problem on 32 bit Machines

Lemma 2.2.2 says we will have lots of primes to use. However, in an early implementation where we chose 16 bit or smaller machine primes on a 32 bit machine, the following problem arose. If we choose the 11^{th} cyclotomic polynomial

$$m(z) = \Phi_{11}(z) = z^{10} + z^9 + z^8 + z^7 + z^6 + z^5 + z^4 + z^3 + z^2 + z + 1$$

to be our minimal polynomial, and a matrix $A \in \mathbb{Z}[z]^{50\times 50}/m(z)$, vector $b \in \mathbb{Z}[z]^{50}/m(z)$ with random 3 decimal digit coefficients to be our input, we ran out of primes since the solution vector x would be a vector of polynomials with coefficient length about 3672 decimal digits long. Only 1 in 11 primes can be used (lemma 2.2.1). If we start with `prevprime(2^16)`, i.e., the largest 16 bit prime, and $m(z) = \Phi_{11}(z)$, the product of all usable primes is 2801 decimal digits long which can only recover fractions with size approximately 2800 decimal digits long. To solve this problem, we use 25 bit floating point primes on 32 bit machines which is supported by Maple.

2.3 Linear p-adic Lifting Approach

2.3.1 Description

The linear p-adic lifting approach is based on developing an integer u in its p-adic representation:

$$u = u_0 + u_1 p + u_2 p^2 + \cdots + u_k p^k$$

where p is an odd positive prime, k is such that $p^{k+1} > 2|u|$, and $u_i \in \mathbb{Z}_p (0 \le i \le k)$.

Consider the polynomial

$$u(z) = \sum_e u_e z^e \in \mathbb{Z}[z]$$

and let p and k be chosen such that $p^{k+1} > 2u_{max}$, where $u_{max} = \max_e |u_e|$. If each integer coefficient u_e is expressed in its p-adic representation

$$u_e = \sum_{i=0}^{k} u_{e,i} p^i \text{ with } u_{e,i} \in \mathbb{Z}_p$$

then the polynomial $u(x)$ can be expressed as

$$u(z) = \sum_{e} (\sum_{i=0}^{k} u_{e,i} p^i) z^e = \sum_{i=0}^{k} (\sum_{e} u_{e,i} z^e) p^i$$

The latter expression for the polynomial $u(z)$ is called a polynomial p-adic representation and its general form is

$$u(z) = u_0(z) + u_1(z)p + u_2(z)p^2 + \cdots + u_k(z)p^k$$

where $u_i(z) \in \mathbb{Z}_p[z]$ for $i = 0, 1, \ldots, k$. The uniqueness of this representation follows from the uniqueness of the p-adic representation of integers.

Definition 2.3.1. Let $a(z) \in \mathbb{Z}[z]$ be a given polynomial. A polynomial $b(z) \in \mathbb{Z}[z]$ is called an order k p-adic approximation to $a(z)$ if

$$a(z) \equiv b(z) \pmod{p^k}$$

The *error*, denoted by e, in approximation $a(z)$ by $b(z)$ is $a(z) - b(z) \in \mathbb{Z}[z]$.

Therefore, $u^{(i)}(z) = u_0(z) + u_1(z)p + \cdots + u_{i-1}(z)p^{i-1}$ is an order i p-adic approximation to $u(z)$.

We wish to utilize the p-adic representation of polynomials with integer coefficients and rational number reconstruction to develop an algorithm to our problem corresponds to Chinese remaindering. In this way, we may use only one prime instead of finding a sequence of primes as we have discussed in Chinese remaindering approach in section 1.3. The purpose of doing this is to reduce the $O(n^3 dL)$ term to to $O(n^3 d)$.

2.3.2 The Subroutines

We will use some of the procedures that we have discussed in section 1.3. These procedures include finding a suitable prime p as well as computing the roots of the minimal polynomial $m(z) \pmod{p}$, polynomial evaluation/interpolation, and rational number reconstruction. In algorithm 2, we avoid doing Chinese remaindering and Gaussian elimination in the main loop. Instead, we compute at the beginning the inverse of input matrix A with respect to all

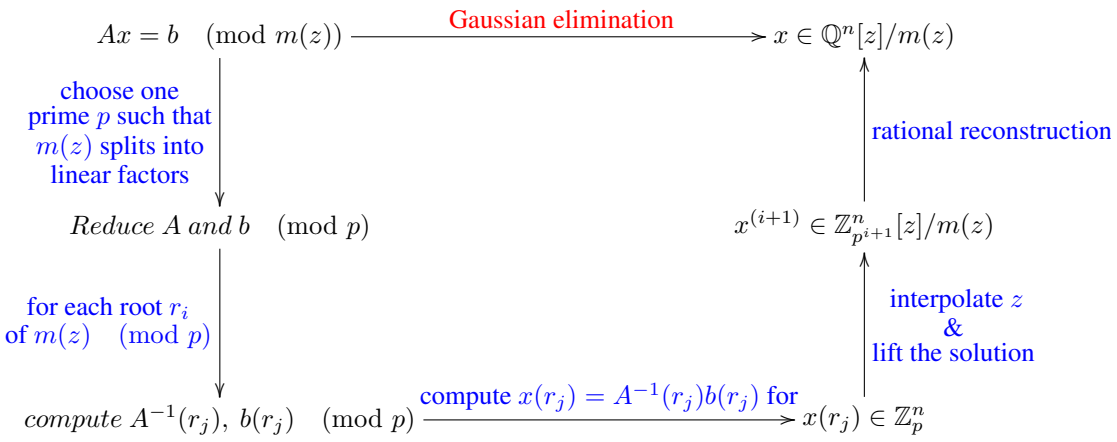

Figure 2.2: Process flow of the linear p-adic lifting approach

the roots of $m(z) \pmod{p}$ and cache them for further computation. In the classical p-adic lifting approach that we discussed in section 1.5, we compute the order $k+1$ approximation from the order k approximation, and keep updating the error term in each iteration until the error term becomes zero. In this problem, we are not able to make the error term zero since the solution vector x may contain fractions. By doing modulo operations mod p^k, we will not be able to get a zero error term unless p is a divisor of the numerator. Therefore, we cannot determine the stopping criteria by just checking if the error goes to zero. So, we must periodically reconstruct $x^{(k)} \in \mathbb{Q}[z]^n \bmod p^k$ using rational reconstruction, and stop when $Ax \equiv b \bmod m(z)$ holds.

Computing the Inverse of Input Matrix $A \pmod{p}$

In algorithm 2, we will need to compute the images of solution vector x by the equation $A(\alpha_i)x(\alpha_i) \equiv e(\alpha_i) \pmod{p}$, where e denotes the updated error term in each iteration. The initial value of e is our input vector b. Its value is updated in each iteration while the input matrix A and the prime p stay the same over all iterations of computation. Therefore, we compute the inverse of $A(\alpha_i)$ in advance so that we just perform matrix vector multiplication in each iteration instead of Gaussian elimination in the main loop. This is the main gain of p-adic lifting over Chinese remaindering. To further reduce the complexity, we use polynomial evaluation to reduce computation to $\bmod p$ and then interpolate the polynomials afterward. Therefore, we will need the inverses of $A(\alpha_i) \pmod{p}$ over all the roots $\alpha_1, \alpha_2, \ldots, \alpha_d$ of

$m(z) \pmod p$. We achieve this in three steps: first reduce the coefficients in A by p; then substitute z by the roots; and lastly compute the inverse of the matrices obtained from the previous subroutine. Gaussian elimination costs $O(n^3)$ operations over $\mathbb{Z}_p$ to compute $A(\alpha_i)^{-1}$.

Updating the Error Term

Similar to the classical p-adic lifting algorithms, we need to update the error term in each iteration. This procedure is simply written as $e_{k+1} = (e_k - Ax_k \bmod m(z))/p$. In section 2.3.4, we use this formula for updating the error term. However, it turns out to be inefficient if we use the straight forward computation. We give two efficient algorithms for updating e_{k+1} in section 2.3.5.

Updating the Image Solution Vector

We need to update the image solution vector x^{k+1} in the k^{th} lifting step before we can use it for rational reconstruction. Similar to the Chinese remainder algorithm that we have discussed in section 2.2.4, we may use either incremental or recursive algorithms for updating the image solution vectors.

2.3.3 The Algorithm

2.3.4 Runtime Analysis of Algorithm 2

We state the running time of Algorithm 2 in terms of n, d, c, and L, the number of lifting steps that Algorithm 2 takes. For p^L to be large enough to reconstruct rationals of in x, $L \in O(\log ||x||_\infty)$.

Proposition 2.3.2. Invert matrix A over all the roots modulo the machine prime p takes $O(n^3d + n^2dc + n^2d^2)$ operations.

Proof. Assuming we have obtained the roots $\alpha_1, \alpha_2, \ldots, \alpha_d$ of $m(z) \pmod p$, we will need to find $A^{-1}(\alpha_1), A^{-1}(\alpha_2), \ldots, A^{-1}(\alpha_d) \in \mathbb{Z}_p^{n\times n}$. As we have described in section 2.2.4, we first reduce the coefficients in the input matrix A and then perform the evaluations and calculate the inverses. The coefficient reduction step takes $O(n^2dc)$ operations (Proposition 2.2.6), and evaluation step takes $O(n^2d^2)$ operations (Proposition 2.2.7). The computation of

Algorithm 2 Algorithm for the **Linear p-adic Lifting Approach**

Input: $A \in \mathbb{Z}[z]^{n\times n}/m(z)$, $b \in \mathbb{Z}[z]^n/m(z)$, $m(z) \in \mathbb{Z}[z]$, $\det(A) \not\equiv 0 \pmod{m(z)}$

Output: $x \in \mathbb{Q}[z]^n$ which satisfies $Ax = b \pmod{m(z)}$

1: Find a new machine prime p s.t. $m(z)$ splits linearly over $\mathbb{Z}_p$, and compute the roots $\alpha_1, .., \alpha_d$ of $m(z) \bmod p$
2: Set $e_0 = b$, $X = 0$.
3: Invert $A(\alpha_i) \bmod p$ for all roots.
if $A^{-1}(\alpha_i) \pmod p$ does not exist **then** Goto 1 **end if**
4: **for** $k = 0, 1, 2, \ldots$ **do**
5: Reduce $e_k \bmod p$.
6: **for** $i = 1$ to d **do**
7: Substitute α_i into e_k.
8: Set $x_k(\alpha_i) = A(\alpha_i)^{-1} e_k(\alpha) \bmod p$.
9: **end for**
10: Interpolate x_k using x_{k1}, ...,x_{kd} wrt. $\alpha_1, .., \alpha_d$.
11: Set $e_{k+1} = (e_k - Ax_k \bmod m(z)) \ / \ p$.
12: Set $X = X + x_k \times p^k$.
13: **if** $k \in \{1, 2, 4, 8, 16, \ldots\}$ **then**
14: Let x be the output of applying rational reconstruction to $X \bmod p^{k+1}$.
15: **if** rational reconstruction succeeds **and** $m(z) | Ax - b$ **then** Output x **end if**
16: **end if**
17: **end for**

inverting the matrices over $\mathbb{Z}_p^{n\times n}$ costs no more than d times the cost of Gaussian elimination therefore has complexity $O(n^3 d)$. Therefore, its total runtime complexity becomes $O(n^3 d + n^2 dc + n^2 d^2)$. □

Before we determine the cost of computing the error e_{k+1} in step 11, we show that $||e_k||_\infty$ is bounded.

Lemma 2.3.3. Let $m(z) = z^d + \sum_{i=0}^{d-1} a_i z^i$ with $a_i \in \mathbb{Z}$. Let $f(z) = \sum_{i=0}^{l} b_i z^i$ with $b_i \in \mathbb{Z}$. Let r be the remainder of f divided m. Then $r \in \mathbb{Z}[x]$ (because m is monic) and $||r||_\infty \leq (1 + ||m||_\infty)^\delta ||f||_\infty$ where $\delta = l - d + 1$.

Proof. The quotient of f divided m has degree $l - d$, hence, there are at most $l - d + 1 = \delta$ subtractions in the division algorithm. The first subtraction is $f_1 := f - b_l x^{l-d} m$. We have $||b_l m||_\infty \leq ||f||_\infty ||m||_\infty$, hence,

$$||f_1||_\infty \leq ||f||_\infty + ||m||_\infty ||f||_\infty = (1 + ||m||_\infty)||f||_\infty.$$

For the purpose of bounding $||r||_\infty$ we assume $\deg f_1 = l - 1$. The next subtraction is $f_2 := f_1 - \mathrm{lc}(f_1) x^{l-1-d} m$. Bounding $|\mathrm{lc}(f_1)| \leq ||f_1||_\infty$ we have

$$||f_2||_\infty \leq ||f_1||_\infty + ||f_1||_\infty ||m||_\infty = (1 + ||m||_\infty)^2 ||f||_\infty.$$

Repeating this argument the result is obtained. □

Theorem 2.3.4. Let e_k be the error term in the k^{th} iteration of Algorithm 2. The absolute value of the integer coefficients in e_k is bounded by $||e_k||_\infty \leq ||[A|b]||_\infty nd(1 + ||m||_\infty)^{d-1}$.

Proof. Given the formula $e_{k+1} = \frac{e_k - Ax_k}{p}$, the initial value $e_0 = b$, and $||x_k||_\infty < p$ for all $k \in \mathbb{Z}$. Let c be the bit length of the maximum of the absolute value of the coefficients in the input matrix A and vector b, i.e., $c = \log_2 \max(||A||_\infty, ||b||_\infty)$. We consider firstly the coefficients in $e_1 = \frac{e_0 - Ax_0}{p}$. The matrix vector multiplication Ax_0 would produce maximum coefficient $||A||_\infty (p-1)nd$. After reducing the polynomials by $m(z)$, the maximum possible coefficient in $Ax_0 \bmod m(z)$ is bounded by

$$||A||_\infty (p-1)nd(1 + ||m||_\infty)^{d-1} \leq 2^c (p-1)nd(1 + ||m||_\infty)^{d-1}$$

by lemma 2.3.3. After subtracting by by e_0 and dividing by p, we know the maximum coefficient in e_1 is bounded by $2^c nd(1 + ||m||_\infty)^{d-1}$. Induction Hypothesis: Assume that

$||e_k||_\infty$ is bounded by $2^c nd(1 + ||m||_\infty)^{d-1}$. Now we know that

$$||e_{k+1}||_\infty \leq \frac{2^c(p-1)nd(1+||m||_\infty)^{d-1} + 2^c nd(1+||m||_\infty)^{d-1}}{p} = 2^c nd(1+||m||_\infty)^{d-1}$$

for every $k \in \mathbb{Z}$. Therefore, we know the bit length of the integer coefficients in e_k is

$$\log ||e_k||_\infty \leq c + \log nd + (d-1)\log(1+||m||_\infty)$$

which is bounded by $O(c+d)$ assuming $||m||_\infty$ is a constant that is smaller than our base B and also $B > nd$. □

Proposition 2.3.5. The runtime complexity of solving the system $Ax_k \equiv e_k \pmod p$ for x_k is $O(n^2 d + d^2)$ using the precomputed inverses of $A(\alpha_i)$, $1 \leq i \leq d$, obtained from Algorithm 3.

Proof. First of all, we reduce the coefficient of e_k modulo p, and then do the polynomial evaluations over the roots. The reduction step takes $O(c+d)$ operations because the length of the coefficients in e_k is bounded by $O(c+d)$, and p is a fixed size machine prime. Each polynomial evaluation takes $O(d)$ operations, and each system solving of $A(\alpha_i)x_k(\alpha_i) \equiv e_k(\alpha_i) \pmod p$, for $x_k(\alpha_i)$ takes $O(n^2)$ operations since $A^{-1}(\alpha_i)$'s have been previously computed. The last step is to do polynomial interpolation to construct $x_k \in \mathbb{Z}_p[z]^n$ which costs $O(nd^2)$ operations. Therefore, the runtime complexity of solving the system $Ax_k \equiv e_k \pmod p$ for x_k in the k^{th} iteration is $O(n^2 d + nd^2)$. □

Proposition 2.3.6. Updating the error term takes $O(n^2 d^2 c)$ operations in each iteration assuming classical polynomial multiplication and division are used for $Ax_k \bmod m(z)$.

Proof. Theorem 2.3.4 gives us an upper bound of the coefficients in the error terms e_k. Therefore, we know that the coefficients in e_k do not grow over the iterations. To update the error term e_{k+1} in the k^{th} iteration, we need to do a matrix vector multiplication of polynomials over $\mathbb{Z}$ then divide by $m(z)$. Assuming $||m(z)||_\infty$ is constant, the matrix vector multiplication costs $O(n^2)$ operations, and the polynomial multiplications contribute another $O(d^2 c)$ factor since the coefficient length in A is bounded by c. The cost of updating the error term is dominated by the above matrix vector multiplication, which is $O(n^2 d^2 c)$. □

Note, In section 2.3.5, two approaches other than classical polynomial multiplication of A and x_k over the polynomials are introduced which reduce the runtime complexity of updating the error term.

Proposition 2.3.7. The runtime complexity of updating the image solution vectors $X^{(L)}$ is $O(ndL^2)$ if we update the solution vector incrementally, and $O(ndM(L)\log L)$ if we update the solution vector recursively, where $M(L)$ is the cost of multiplying integers of length L.

Proof. Similar to the Chinese remaindering procedure that we have discussed in section 2.2.4, updating the image solutions incrementally causes multiplications between small integers, e.g., coefficients of $x_k \in \mathbb{Z}_p[z]^n$, and big integers, e.g., p^k, in each iteration. Therefore, faster integer multiplication algorithms do not apply. As a result, the complexity of incremental updating becomes $\sum_{i=1}^{L} ndi \in O(ndL^2)$. In the case of recursive updating, we update the solution vector x_k as follows: $x_0 + x_1p + x_2p^2 + \cdots + x_kp^k = (x_0 + x_1p + x_2p^2 + \cdots + x_{\frac{k}{2}-1}p^{\frac{k}{2}-1}) + p^{\frac{k}{2}}(x_{\frac{k}{2}} + x_{\frac{k}{2}+1}p + x_{\frac{k}{2}+2}p^2 + \cdots + x_kp^{\frac{k}{2}})$. There are $\log L$ levels of recursion, and in the i^{th} recursion, the cost is $O(ndM(\frac{L}{2^i})L) \in O(ndM(L))$ where $M(L)$ is the cost of fast integer multiplication which is usually $M(N) = N\log N\log\log N$. Therefore, the runtime complexity becomes $O(ndM(L)\log L)$ if it is done using fast integer multiplication, e.g. FFT for big integer multiplication. □

Theorem 2.3.8. The total running time of the p-adic lifting approach is $O(n^3d + n^2d^2cL + ndL^2)$ if we use classical polynomial and integer algorithms.

The first contribution, n^3d, is the cost of the d matrix inversions. The second, n^2d^2cL, is the total cost of computing the error terms e_k and the trial division $m(z)|Ax-b$. The third, ndL^2, is the cost of converting the solution vector to integer polynomial representation from its p-adic representation.

Proof. In this algorithm, we only need one prime p such that the minimal polynomial splits linearly over $\mathbb{Z}_p$. The time for computing this can be ignored. In step 3, we pre-compute the inverse of the matrix A at each root modulo p using Gaussian elimination. This costs $O(n^3d)$ arithmetic operations in total. Step 5 costs $O(ndcL + nd^2L)$ operations since e_k is a vector of polynomials of degree $< d$ with coefficient length in $O(c+d)$ and is done for L iterations. The substitution of all d roots into e_k costs $O(nd^2L)$ operations. Computing the solution vector $x_k(\alpha_i)$ is just a matrix vector multiplication modulo p which costs $O(n^2dL)$ in total. Interpolation costs $O(nd^2L)$ which is the same as in algorithm 1. To compute the error e_k

in step 11, we should do a matrix vector multiplication of polynomials over $\mathbb{Z}$ then divide by $m(z)$. The cost is dominated by this computation which is $O(n^2d^2cL)$ operations since fast integer multiplication is not applicable here when using classical polynomial multiplication. The cost of adding x_kp^k to X is $O(ndL^2)$ which is the same cost as rational reconstruction in both algorithms 1 and 2. Trial division in step 15 costs $O(n^2d^2cL)$ operations which is the same cost of computing the error term. Therefore, the total running time for algorithm 2 is $O(n^3d + n^2d^2cL + ndL^2)$. □

2.3.5 Computing the Error Term

In our implementation of Algorithm 2, the most expensive component is the computation of the error term in step 11. In particular, the matrix vector multiplication Ax_k needs to be computed over $\mathbb{Z}$. This requires n^2 polynomial multiplications. Assuming classical polynomial multiplication and integer multiplication, it has complexity $O(n^2d^2c)$ for each iteration. We consider two approaches which theoretically reduce the runtime complexity by a factor of d.

Without loss of generality, let $C = Ax \bmod m(z) \in \mathbb{Z}[z]^n$, where x represents x_k in the k^{th} iteration. From Theorem 2.3.4, we learn that $||C||_\infty \leq ndp||A||_\infty(1+||m||_\infty)^{d-1}$. We discuss below two approaches, namely "pre-CRT" and "Single-Point Evaluation/Interpolation", to reduce the complexity of computing C. Note that, since the length of the vector C is more than $O(n^2dc)$ in general, we may not expect to reduce the complexity of computing the error term by more than a factor of d.

The pre-CRT Approach of Computing Ax_k

As we have tried in section 2.2, we can transfer the operations over polynomials into the computations over integers mod p, hence reduce the complexity. To compute $C = Ax \bmod m(z)$, We pick a sequence of machine primes $p_1, p_2, p_3, \ldots$ such that $m(z)$ splits into distinct linear factors over $\mathbb{Z}_{p_i}$. For each prime p_i, we find the roots $\alpha_1, \alpha_2, \ldots, \alpha_d$ of $m(z) \bmod p_i$ and substitute them into A and x, then interpolate over z from $A(\alpha_i)x(\alpha_i), \alpha_i$ to obtain $C_i \in \mathbb{Z}_{p_i}[z]^n$. After we apply the above procedure for sufficiently many primes, we can use the Chinese remainder algorithm to obtain $C \in \mathbb{Z}_{p_1 \times p_2 \times p_3 \times \cdots}[z]^n$, which is the same as $C \in \mathbb{Z}[z]^n/m(z)$. For example, let α_j be one of the d roots of $m(z) \pmod{p_i}$. We compute $C_i(\alpha_j) = A(\alpha_j)x_k(\alpha_j) \bmod p_i$ for all d roots, hence obtain $C_i \in \mathbb{Z}_{p_i}[z]$ by interpolating the

pairs $(\alpha_1, C_i(\alpha_1)), (\alpha_2, C_i(\alpha_2)), \ldots, (\alpha_d, C_i(\alpha_d))$ over z. The Chinese remainder algorithm is applied to obtain $C \in \mathbb{Z}[z]^n$ in the last step. We may use either incremental CRT or recursive CRT as described in section 2.2.4.

Algorithm 3 Pre-calculation of pre-CRT Algorithm in Computing the Error Term

Input: $A \in \mathbb{Z}[z]^{n\times n}/m(z)$, $m(z) \in \mathbb{Z}[z]$
Output: void

1: Find primes $p_1, p_2, \ldots, p_t$ such that $\prod p_i > 2||C||_\infty$ and $m(z)$ splits into linear factors over $\mathbb{Z}_{p_i}$ for $1 \leq i \leq t$
2: **for** $i = 1$ to t **do**
3: Set $A_i = A \bmod p_i$
4: Find all roots $\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{id}$ of $m(z)$ in $\mathbb{Z}_{p_i}$
5: **for** $j = 1$ to d **do**
6: Set $A_{ij} = A_i(\alpha_j) \in \mathbb{Z}_{p_i}^{n\times n}$
7: **end for**
8: **end for**

Algorithm 4 Main Steps of pre-CRT Algorithm in Computing the Error Term

Input: $A \in \mathbb{Z}[z]^{n\times n}/m(z)$, $x \in \mathbb{Z}_p[z]^n$, $m(z)$, p_i, A_{ij},for $1 \leq i \leq t, 1 \leq j \leq d$
Output: $C = Ax \pmod{m(z)} \in \mathbb{Z}[z]^n$

1: **for** $i = 1$ to t **do**
2: **for** $j = 1$ to d **do**
3: Compute $x_{ij} = x(\alpha_j) \in \mathbb{Z}_{p_i}^n$
4: Set $C_i(\alpha_j) = A_i(\alpha_j)x(\alpha_j) \pmod{p_i}$
5: **end for**
6: Interpolate $C_i(z) \in \mathbb{Z}_p[z]^n$ from pairs $(\alpha_1, C_i(\alpha_1)), (\alpha_2, C_i(\alpha_2)), \ldots, (\alpha_d, C_i(\alpha_d))$
7: **end for**
8: Apply Chinese remaindering to recover C from $C_1(z), C_2(z), \ldots, C_t(z)$ and $p_1, p_2, \ldots, p_t$

The above approach would give an even worse runtime complexity than what is stated in Proposition 2.3.6 if we had to use different primes in each lifting step (for each k). However, we may use the same sequence of primes for each iteration, hence we may pre-compute $A(\alpha_i)$'s in advance to speed up the computation. Now the question is how many such primes do we need? In the beginning of section 2.3.5, we have shown that there is an upper bound on the integer coefficients in $C = Ax \bmod m(z)$. Therefore, we know that there is a fixed number of primes which are needed in order to recover the coefficients in C. By using the bound, we can compute in advance a sequence of suitable primes and their roots along with $A(\alpha)$'s which will be used repeatedly in each iteration.

Single-Point Evaluation/Interpolation Approach of Computing Ax_k

We adopt the notations that have been used for the pre-CRT approach. We know that $||C||_\infty$ has a bound and we may use that bound to decide an appropriate integer for the single-point evaluation/interpolation method which is described in section 1.6.1. Let $C(z) = Ax \bmod m(z)$. It is sufficient to choose $\zeta \geq 2||C||_\infty$. Upon finding $\zeta \in \mathbb{Z}$, we can substitute ζ into A, x, and m, and compute $C(\zeta) = A(\zeta)x(\zeta) \bmod m(\zeta)$ by a matrix vector multiplication over $\mathbb{Z}$ followed by integer divisions. Finally, we may do single-point interpolation based on $C(\zeta) \in \mathbb{Z}^n_{m(\zeta)}$ to obtain $C(z) \in \mathbb{Z}[z]^n/m(z)$.

Runtime Complexity of pre-CRT and Single-Point Evaluation/Interpolation algorithms

The number of primes needed in the pre-CRT algorithm is $O(c+d)$ which is determined by the bound $2||C||_\infty$. In the pre-calculation procedure, the complexities of finding the primes and calculating the roots are dominated by later steps (see section 2.2.4). For each prime p, reducing the coefficients in A costs $O(n^2dc)$ operations over $\mathbb{Z}_p$; evaluating A_p over d roots costs $O(n^2d^2)$ operations. Therefore, the overall cost of computing the cached items by algorithm 3 is $O(n^2d^2c+n^2d^3+n^2dc^2)$. In the main loop of linear p-adic lifting, algorithm 4 is used to compute Ax. For each prime, it costs $O(nd^2)$ operations to evaluate x over d roots, it costs $O(n^2d)$ operations to compute $C_j(\alpha)$'s by integer matrix vector multiplications over $\mathbb{Z}_p$, and $O(nd^2)$ operations to do polynomial interpolations to obtain C. Therefore, the overall cost of algorithm 4 is $O(nd^2c + n^2dc + n^2d^2 + nd^3 + n^2dc^2)$.

The cost of single-point evaluation/interpolation algorithm is dominated by the cost of big integer multiplications in $A(\zeta)x(\zeta)$. This is achieved by choosing our evaluation point $\alpha > 2||C||_\infty$ such that α is a power of 2. Thus the evaluations and interpolations can be done in $O(d \log \alpha)$ operations. Then we need to compute $C(\alpha) = A(\alpha)x(\alpha)$ which involve multiplications between integers with length $O(d \log \alpha)$. Therefore, the cost of single-point evaluation/interpolation is $O(n^2 M(d \log \alpha))$, where $M(e)$ is the cost of multiplying integers of length e, hence $\tilde{O}(n^2(cd + d^2))$ since $\log \alpha \in O(c + d)$ and fast integer multiplication algorithms are used.

2.3.6 Attempt at a Quadratic p-adic Lifting Approach

We have also designed, implemented, and analyzed a quadratic p-adic lifting approach to solve linear systems of equations over cyclotomic fields. However, from both the analysis and the timing, it turns out to be worse than both the Chinese remaindering approach and the linear p-adic lifting approach are. The bottleneck in the quadratic p-adic lifting algorithm is the computation of the error term and solving $Ax_k = b \pmod{p^{2^k}}$, for x_k in the k^{th} iteration, which can not be reduced by using modulo techniques since it lifts the coefficients in the solution vector from $p^{2^{k-1}}$to p^{2^k} in the k^{th} iteration.

2.4 An Upper Bound of the Coefficients in the Solution Vector x

Our solution vector $x \in \mathbb{Q}[z]^n$ can have large fractions. In this section, we determine a bound for their size.

2.4.1 The Hadamard Maximum Determinant Problem

Given a matrix $A \in \mathbb{Q}^{n \times n}$, the Hadamard maximum determinant problem is to find the largest possible determinant of A. Hadamard proved that the determinant of any complex $n \times n$ matrix A with entries in the closed unit disk $|a_{ij}| \leq 1$ satisfies $|det(A)| \leq n^{\frac{n}{2}}$. Here we only need an upper bound of the determinant which is called the Hadamard bound:

$$\det(\mathrm{A}) \leq \sqrt{\prod_{\mathrm{i}=1}^{\mathrm{n}} \sum_{\mathrm{j}=1}^{\mathrm{n}} \mathrm{A}_{\mathrm{ij}}^2} \tag{2.1}$$

If $c = \max_{i,j} |A_{ij}|$, then we get $\det(A) \leq n^{\frac{n}{2}} c^n$.

2.4.2 A Hadamard-Type Bound on the Coefficients of a Determinant of a Matrix of Polynomials

Goldstein and Graham discussed the problem "Hadamard-Type Bound on the Coefficients of a Determinant of Polynomials" [6] and gave the following result:

Lemma 2.4.1 (Goldstein and Graham, 1974)**.** Let A be an n by n matrix of polynomials in $\mathbb{Z}[z]$. Let A' be the matrix of integers with $A'_{i,j} = ||A_{i,j}||_1$ that is, $A'_{i,j}$ is the one norm of $A_{i,j}$. Let H be Hadamard's bound for $\det A'$. Then $||\det A||_\infty \leq H$.

Since $\deg_z A_{i,j} \leq d-1$ we have $A'_{i,j} \leq d||A||_\infty$. Applying Hadamard's bound to bound $|\det A'|$ we obtain

$$||\det A||_\infty \leq \prod_{i=1}^{n} \sqrt{\sum_{j=1}^{n} A'^2_{i,j}} = d^n n^{n/2} ||A||_\infty^n \leq d^n n^{n/2} 2^{cn}.$$

To calculate $\mathrm{res}_z(\det A, m(z))$, because $m(z)$ is monic

$$\mathrm{res}_z(\det A, m(z)) = \pm \mathrm{res}(r(z), m(z))$$

where $r(z)$ is the remainder of $\det A$ divided $m(z)$. Applying Lemma 2.3.3 to determine $||r||_\infty$ we have $\deg_z \det A \leq n(d-1)$ thus $\delta \leq n(d-1) - d + 1 = (n-1)(d-1)$ and

$$||r||_\infty \leq (1 + ||m||_\infty)^{(n-1)(d-1)} d^n n^{n/2} ||A||_\infty^n.$$

Let $R = \mathrm{res}_z(r(z), m(z))$. Note that R is an integer. To bound $|R|$ recall that $R = \det S$ where S is Sylvester's matrix for the polynomials $r(z)$ and $m(z)$. Now $\deg_z r < d$ but for the purpose of bounding $|R|$ we assume $\deg_z r = d-1$. Then S is a $2d-1$ by $2d-1$ matrix of integers where the d coefficients of $r(z)$ are repeated in the first d rows of S and the $d+1$ coefficients of $m(z)$ are repeated in the last $d-1$ rows. Applying Hadamard's bound to the rows of S we obtain

$$|\det S| \leq \sqrt{d||r||_\infty^2}^d \times \sqrt{(d+1)||m||_\infty^2}^{d-1}$$

from which we obtain the following result where we used $\sqrt{d+1}^{d-1} < \sqrt{d}^d$ for $d > 1$ to simplify the result.

Theorem 2.4.2. The length of the maximum absolute value of the coefficients in the output vector $x \in \mathbb{Q}[z]^n$ produced by Algorithm 1 and 2 which satisfies $Ax = b \pmod{m(z)}$ is bounded by $O(ndc + nd^2)$ assuming $||m||_\infty$ is bounded by the base B, that is,

$$\log ||x||_\infty \in O(ndc + nd^2).$$

Proof. Let $R = \mathrm{res}_z(\det r(z), m(z))$. Then

$$|R| < d^{nd+d} ||m||_\infty^{d-1} (1 + ||m||_\infty)^{(n-1)(d-1)d} n^{dn/2} ||A||_\infty^{nd}.$$

This means the size of the denominators in $x = A^{-1}b$ can be more than nd times longer than $||A||_\infty$. Recall the equation

$$Cf + Dg = \mathrm{res}_z(f, g)$$

where C and D are polynomials with integer coefficients and $\deg C < \deg g, \deg D < \deg f$. We replace f by $m(z)$, and replace g by $\det A$ then get that $D/\mathrm{res}_z(f,g)$ is the inverse of $\det A \bmod m(z)$. Because of the fact that the integer coefficients in D can be found among the determinants of the minors of Sylvester matrix $\mathrm{Syl}_z(r(z), m(z))$, we can obtain an upper bound for $||D||_\infty$, that is,

$$||D||_\infty < d^{nd+d}||m||_\infty^{d-1}(1+||m||_\infty)^{(n-1)(d-1)d}n^{dn/2}||A||_\infty^{nd}.$$

By Cramer's rule, we know that

$$x_i = \frac{\det A^{(i)}}{\det A} \bmod m(z).$$

Since we have assumed that $c = \log_2 \max(||A||_\infty, ||b||_\infty)$, we can use the bound we obtained for $\det A \bmod m(z)$ to bound $r^{(i)} = \det A^{(i)} \bmod m(z)$, that is,

$$||r^{(i)}||_\infty \leq (1+||m||_\infty)^{(n-1)(d-1)}d^n n^{n/2}2^{cn}.$$

The last step is to obtain $x_i = r^{(i)}\frac{D}{R} \bmod m(z) = (r^{(i)}D \bmod m(z))/R$ from which we can obtain the bound

$$||x_i||_\infty \leq (1+||m||_\infty)^{d-1}d||r^{(i)}||_\infty||\frac{D}{R}||_\infty.$$

Hence, we can now conclude that $\log||x||_\infty \in O(ndc+nd^2)$ because $\log||r^{(i)}||_\infty$ is in $O(nd+nc)$ and $\log||\frac{D}{R}||_\infty$ is in $O(ndc+nd^2)$ and the rest terms are dominated by these terms for i from 1 to n. □

This bound may be used to bound the number of primes needed in Algorithm 1, and the number of lifting steps in Algorithm 2 to output x while the input is a non-singular system. However, in our experiments on systems given by Vahid Dabbaghian (Table 2.2,2.3), the number of primes (lifting steps) used are much smaller than the bound.

2.5 Runtime Complexity Comparison

We have shown that the runtime complexity of the Chinese remaindering approach is $T_{crt} = O(n^3dL + n^2d^2cL + ndL^2)$ where L is the number of machine primes used, and the runtime complexity of the p-adic lifting approach is $T_{lift} = O(n^3d + n^2d^2cL + ndL^2)$ where L is the number of lifting steps. In section 2.4 we showed for $||m||_\infty < B$ that $L \in O(ndc+nd^2)$. We may now compare the runtime complexities of Algorithm 1 and 2 just in terms of the

variables n, d, and c in the input. Therefore, T_{crt} becomes $O(n^4d^2c + n^3d^3c^2 + n^4d^3 + n^3d^4c + n^3d^5)$, and T_{lift} becomes $O(n^3d^3c^2 + n^3d^4c + n^3d^5)$. The n^4d^2c and n^4d^3 terms in T_{crt} are contributed by the Gaussian eliminations over $\mathbb{Z}_p$. The $n^3d^3c^2$, n^3d^4c and n^3d^5 terms are contributed by the rational reconstruction and trial division of $m(z)|Ax - b$ in both Algorithm 1 and 2, and it also represents the contribution by updating the error terms and adding up the image solution vectors in Algorithm 2.

The pre-CRT and single-point evaluation/interpolation algorithms are used in Algorithm 2 to improve the complexity of computing the error term. However, they do not change the overall complexity of Algorithm 2 even though we observe a better running time in tables 2.1, 2.2, and 2.3. Fast algorithms are used to do Chinese remaindering in Algorithm 1, adding up the image solution vectors in Algorithm 2, and rational reconstructions in both algorithms. However, they do not change the runtime complexity of either Algorithm 1 or Algorithm 2.

2.6 Implementation and Timings

We have implemented Algorithms 1 and 2 in Maple 10. Both of the algorithms are output sensitive. In our programs, we used the Maple library routines `iratrecon` for rational number reconstruction, and our own routine `iscyclotomic` to find the order k of the given cyclotomic polynomial. We used the library routine `Roots(m) mod p` to find the roots of $m(z)$ in $\mathbb{Z}_p$. We use 25 bit floating point primes on 32 bit machines, and 31 bit integer primes on 64 bit machines. If we choose the primes as stated, we can take advantage of the fast C code in the `LinearAlgebra:-Modular` package which provides fast polynomial evaluation, linear solving and matrix inversion over $\mathbb{Z}_p$. We implemented the recursive versions of Chinese remainder algorithm and updating the solution vectors in the linear p-adic lifting approach where we use pre-CRT to compute the error terms.

2.6.1 Timing the Random Systems and Real Systems

We chose a set of randomly generated systems and two sets of real systems given by Dr. Vahid Dabbaghian-Abdoly for our benchmarks. All timings we give in the following were obtained using Maple 10 on an AMD® Opteron 150 processor @ 2.4 GHz with 12GB of RAM. Our programs are designed for dense inputs. They do not take advantage of any

structure if the input systems are sparse.

Data Set 1:
For the first data set we use the $7th$ cyclotomic polynomial $m(z) = 1 + z + z^2 + z^3 + z^4 + z^5 + z^6$ as the minimal polynomial. The first data set consists of systems of dimension $5, 10, 20, 40, 80, 160$ where the entries of A and b were generated using the Maple command

```
> f := randpoly(z,dense,degree=5,coeffs=rand(2^c)):
```

for different values of c which specifies the lengths of the integer coefficients in binary digits. This Maple command outputs a dense polynomial in z with degree 5 and coefficients uniformly chosen at random from $[0, 2^c)$.

Table 2.1 shows the running time of dense random polynomial inputs for both of our algorithms, namely "CRT" and "Lift". In addition, the pre-CRT and single-point evaluation/interpolation algorithms embedded in the linear p-adic lifting algorithm are timed, namely "Lift1" and "Lift2". We timed Gaussian elimination, namely "GE", as a complement by using an optimized version of Gaussian elimination written in Maple by Dr. Michael Monagan. This procedure gives a much better running time than the Maple `LinearSolve` procedure even though they both use Gaussian elimination hence with same complexity. We observe that either of our improved linear p-adic lifting approaches are much faster than the Chinese remaindering approach when the dimension n of the input matrix and input vector gets larger, and our modular algorithms beats Gaussian elimination when the dimension of the matrix is bigger than 20.

Remark: In all of our timings, we write the runtime in CPU seconds.
Data Set 2:
The problems in this data set were given to us by Vahid Dabbaghian. They include systems with various dimensions, coefficient lengths, and minimal polynomials. The systems are available at

`http://www.cecm.sfu.ca/CAG/code/VahidsSystems.zip`

Table 2.2 and Table 2.3 show the running times for the systems given to us by Vahid Dabbaghian. The labeling of the algorithms is the same as in Table 2.1. In addition, we

show here the number of machine primes that are needed to construct the solution vector in the Chinese remaindering approach. This corresponds to the number of lifting steps needed in the linear p-adic lifting algorithms. One can see that the modular algorithms are much faster than Gaussian elimination.

n	Coefficient length c in binary digits										Remark
	2	4	8	16	32	64	128	256	512	1024	
5	.022	.037	.029	.061	.085	.139	.311	.789	2.308	7.396	GE
	.029	.029	.062	.149	.385	.770	1.717	4.122	10.66	31.30	CRT
	.027	.026	.051	.104	.263	.655	1.899	6.018	22.53	85.79	Lift1
	.020	.019	.051	.083	.142	.328	.725	1.854	5.536	19.84	Lift2
	.024	.024	.044	.102	.196	.576	1.176	2.683	7.012	19.15	Lift
10	.108	.167	.222	.349	.637	1.303	3.150	8.814	27.27	93.62	GE
	.046	.096	.223	.498	1.242	2.823	6.981	18.47	51.92	158.8	CRT
	.047	.079	.186	.318	.834	2.198	6.760	22.39	86.32	306.5	Lift 1
	.052	.075	.122	.253	.550	1.382	3.447	8.577	26.84	96.57	Lift2
	.054	.118	.231	.451	.956	3.260	6.981	16.22	43.55	118.4	Lift
20	.865	1.290	2.013	3.564	7.172	16.36	43.14	127.8	421.6	1507	GE
	.202	.414	.868	1.892	5.762	14.70	37.34	104.2	310.8	930.3	CRT
	.156	.305	.616	1.233	3.327	8.474	26.59	89.53	325.6	1236	Lift1
	.154	.249	.472	1.010	2.486	6.052	16.08	46.48	154.4	584.8	Lift2
	.314	.698	1.442	2.981	6.358	23.20	50.47	115.0	310.8	827.4	Lift
40	9.593	14.58	24.45	46.96	102.6	259.1	745.4	2402	–	–	GE
	.877	2.034	4.252	9.412	32.88	86.82	226.0	644.1	2038	–	CRT
	.571	1.086	2.325	5.311	12.91	34.06	108.7	368.6	1427	–	Lift1
	.635	1.273	2.568	5.281	13.68	33.57	92.23	285.1	942.7	4047	Lift2
	2.104	4.706	10.08	21.67	46.85	190.7	411.8	916.7	2415	–	Lift
80	152.2	236.5	380.4	820.1	1925	–	–	–	–	–	GE
	11.44	15.13	32.54	71.59	250.8	659.3	1780	–	–	–	CRT
	2.385	4.150	8.801	28.55	60.77	166.1	550.8	1915	–	–	Lift1
	3.594	6.530	13.85	29.46	79.44	188.4	550.2	1777	–	–	Lift2
	16.04	36.36	79.89	177.4	383.0	1526	3203	–	–	–	Lift
160	2777	4414	–	–	–	–	–	–	–	–	GE
	91.23	120.9	264.8	581.1	2283	–	–	–	–	–	CRT
	17.90	19.08	38.87	107.3	264.5	784.0	2733	–	–	–	Lift1
	45.81	46.26	93.94	207.1	560.7	1364	4140	–	–	–	Lift2
	251.4	318.3	706.8	1650	3528	–	–	–	–	–	Lift
$m(z) := z^6 + z^5 + z^4 + z^3 + z^2 + z + 1$, $d = 6$											

Table 2.1: Runtime (in CPU seconds) of Random dense input with various dimensions and coefficients. "–" denotes the running time is over 5,000 seconds.

file	sys49	sys100	sys100b	sys144	sys196	sys225	sys256	sys576	sys900	sys900b
$deg_z(m)$	4	8	4	2	2	4	4	6	8	2
k	5	24	8	4	3	5	12	7	24	4
$\log \|\|A\|\|_\infty$	10	5	2	4	11	2	3	3	2	5
$\log \|\|x\|\|_\infty$	45	14	1	1	229	875	2	1	2	1
CRT	.053	.185	.034	.050	1.327	3.462	.258	1.304	3.223	2.182
Lift 1	.115	.381	.123	.203	1.985	1.810	.848	3.972	8.710	5.995
Lift 2	.095	.296	.098	.160	2.596	1.593	.621	2.818	7.191	5.848
GE	3.956	165.3	.882	.282	221.8	23.60	4.988	13.17	19.86	13.84
# primes	2	1	1	1	16	32	1	1	1	1

Table 2.2: Runtime (in CPU seconds) on some of the systems given by Vahid Dabbaghian.

file	144Huge	196Huge	256Huge	256Huge2	256Huge3	324Huge	400Huge	484Huge
$deg_z(m)$	40	12	4	24	24	16	108	88
k	55	13	12	39	39	17	171	276
$\log \|\|A\|\|_\infty$	83	57	596	129	59	196	707	808
$\log \|\|x\|\|_\infty$	159	108	90010	255	114	573	2202	2504
CRT	21.19	6.808	3904	64.01	24.99	67.97	12063	13653
Lift 1	22.62	5.850	1908	49.16	22.45	63.41	8536	8632
Lift 2	12.11	3.997	3349	60.68	11.40	60.59	26816	36529
GE	–	18495	3681	10789	–	15036	*	*
# primes	8	8	4096	16	8	16	128	128

Table 2.3: Runtime (in CPU seconds) on some of the huge systems given by Vahid Dabbaghian. "–" denotes the running time is over 50,000 seconds. "*" denotes run out of memory.

Chapter 3

Conclusion

We designed and implemented three modular algorithms to solve linear systems of equations over cyclotomic fields. They use Chinese remaindering, linear p-adic lifting, and quadratic p-adic lifting. All of them use rational number reconstruction. The first two algorithms are presented in this thesis along with a complexity analysis and timings on random and real systems. The timings and analyses show that the modular algorithms are much faster than ordinary Gaussian elimination. From both our timings and runtime analysis, the quadratic p-adic lifting approach is not as efficient as the first two. Therefore, it was not included in this thesis. Both the Chinese remaindering and linear p-adic lifting approaches assume that there are many primes which split $m(z)$ into distinct linear factors and that it is easy to find them. Both of the modular algorithms discussed in this thesis may be modified to solve linear systems of equations over general number fields, provided the minimal polynomial $m(z)$ is monic with integer coefficients and we can easily find primes which split $m(z)$. However, as we have mentioned in lemma 2.2.2, the probability that a prime splits an irreducible polynomial in $\mathbb{Q}[z]$ into distinct linear factors is approximately $1/d!$ in general which severely limits this approach.

In the Chinese remaindering approach, we modulo the input over a sequence of primes and it is clear that we can use parallelism in many places. However, this topic is beyond the scope of this thesis.

Bibliography

[1] Paul Bateman and Harold Diamond. *Analytic Number Theory – An Introductory Course.* World Scientific, 2004.

[2] Zhuliang Chen and Arne Storjohann. A BLAS based C library for exact linear algebra on integer matrices. *Proceedings of ISSAC '05*, ACM Press, pp. 92–99, 2005.

[3] P. S. Wang, M. J. T. Guy, J. H. Davenport. p-adic Reconstruction of Rational Numbers. SIGSAM Bulletin, 16, No 2, 1982.

[4] J. D. Dixon. Exact solution of linear equations using $p-$adic expansions. *Numer. Math.* **40** pp. 137–141, 1982.

[5] R. Moenck and J. Carter. Approximate algorithms to doerive exact solutions to systems of linear equations. *Proceedings of EUROSAM '79*, Springer Verlag LNCS **72**, pp. 65–72, 1979.

[6] A. Goldstein and G. Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review* **1** 394–395, 1974.

[7] Michael Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. *Proceedings of ISSAC '04*, ACM Press, pp. 243–249, 2004.

[8] G. E. Collins and M. J. Encarnacion. Efficient Rational Number Reconstruction. *J. Symbolic Computation* **20**, pp. 287–297, 1995.

[9] Michael Rabin. Probabilistic Algorithms in Finite Fields. *SIAM J. Computing* **9**(2) pp. 273–280, 1980.

[10] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*, University of Cambridge Press, 1999.

[11] K.O. Geddes, S.R. Czapor, G. Labahn. *Algorithms for Computer Algebra*, Kluwer Academic Publishers, 1992.

[12] M.B. Monagan, and G. H. Gonnet. Signature functions for algebraic numbers. *Proceedings of ISSAC '94* ACM Press, New York, NY, 291-296.