Sparse Hensel Lifting Algorithms for Multivariate Polynomial Factorization

by

Tian Chen

M.Sc., University of Ontario Institute of Technology, 2011
B.Sc., Imperial College London, 2008

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2024

Declaration of Committee

Name:	**Tian Chen**
Degree:	**Doctor of Philosophy**
Thesis title:	**Sparse Hensel Lifting Algorithms for Multivariate Polynomial Factorization**
Committee:	**Chair:** Tom Archibald Professor, Mathematics

Michael Monagan
Supervisor
Professor, Mathematics

John Stockie
Committee Member
Professor, Mathematics

Marni Mishna
Examiner
Professor, Mathematics

Mark Giesbrecht
External Examiner
Professor
Mathematics
University of Waterloo

Abstract

Let a be a polynomial in $\mathbb{Z}[x_1, \cdots, x_n]$ that is represented by a black box. In this thesis, we have designed and implemented a new factorization algorithm that, on input of the black box, outputs the irreducible factors of a in the sparse representation. Our new algorithm based on sparse Hensel lifting applies equally well to general multivariate polynomials, both sparse and dense. We first designed the algorithm for a being monic in x_1 and square-free, then completed the factorization problem by considering a being non-monic, non-square-free, and non-primitive. Our algorithm first finds the factors of the primitive part of a, then the factors of the content of a in the main variable x_1. We implemented our algorithm in Maple with some subroutines in C. A variety of timing benchmarks are presented. All our timings are much faster than the current best determinant and factorization algorithms in Maple and Magma. We also present a worst-case complexity analysis of our new black box factorization algorithm, along with a failure probability analysis. The case for large integer coefficients has also been considered.

Keywords: sparse Hensel lifting; multivariate polynomial factorization; black box factorization; determinant computation

Acknowledgements

I would like to give my deepest gratitude to my PhD supervisor, Professor Michael Monagan. He has provided me with academic guidance and invested generously in me with his time and research money. He has encouraged me continually, so my confidence has grown steadily in academic and professional settings. I would not have become who I am today without knowing and working with Prof. Monagan. As one of the leading experts in computer algebra, he inspired me to press on so that I can also be a contributor to the field.

I would like to thank my committee member, Professor John Stockie. Prof. Stockie was initially my PhD supervisor but became unable to supervise me one year after I started the program due to an extenuating circumstance. Without his initial support, I would not have had the opportunity to do my PhD at SFU. While I was searching for a new PhD supervisor, several faculty and staff members in the Math department helped me, including but not limited to Prof. Weiran Sun, Prof. Razvan Fetecau, Prof. David Muraki, Prof. Steven Ruuth, Prof. Nilima Nigam, and our former graduate program assistant Sofia Leposavic.

Let me give thanks to the examiner, Prof. Marni Mishna, and the external examiner, Prof. Mark Giesbrecht, for reading my thesis.

I thank my colleague Garrett Paluck for his bivariate Hensel lifting C code, which I integrated into my code as a subroutine.

Many other SFU colleagues have helped me, as well as other research professionals in computer algebra locally, nationally, and internationally.

Among friends and family members who supported me, I especially thank my parents for their many practical acts of help when I was in need. I thank my dear grandma for her moral support, as she always trusts me.

Most importantly, I thank God for entrusting me with such a challenge. I give glory to my Lord Jesus, for He gives me wisdom and strength to endure a long and difficult journey. My Lord filled me with hope, especially during 2021 and 2023, in which I endured physical and mental suffering. “We also glory in our sufferings, because we know that suffering produces perseverance; perseverance, character; and character, hope. “ – Romans 5:3-4.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Multivariate polynomial factorization: a brief history

Polynomial factorization is one of the central problems in computer algebra. It has many applications in other fields such as algebraic coding theory, cryptography, number theory and algebraic geometry [19]. In this thesis, we consider the following problem (Problem $\mathcal{P}$) of factoring a multivariate polynomial over the ring of integers. Our work focuses on the design, analysis and implementation of algorithms to solve Problem $\mathcal{P}$.

Problem $\mathcal{P}$: Given a polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$, compute the irreducible factors of a with coefficients in $\mathbb{Z}$.

Note that we do not factor the integer content. For example, in $6x^2-6y^2 = 6(x+y)(x-y)$, we will not factor the integer content 6.

One of the main algorithmic tools for solving Problem $\mathcal{P}$ is *sparse Hensel lifting* (SHL). Hensel lifting was initially used for factoring univariate polynomials over the integers [59]. Zassenhaus's algorithm [59], developed in 1969, first factors the polynomial mod p, then Hensel lifts the factors to mod p^{2^k}. For factoring univariate polynomials over finite fields, see [3, 7] for early references. The Cantor-Zassenhaus algorithm from 1981 [7] is a Las Vegas algorithm for factoring a univariate polynomial a prime field $\mathbb{F}_p$. It is implemented in many computer algebra systems including Maple and Magma.

The idea of Hensel lifting can be extended to multivariate polynomials. All multivariate polynomial factorization algorithms reduce to univariate factorization in $\mathbb{Z}[x_1]$. For multivariate polynomial factorization algorithms that use Hensel lifting, Hensel lifting is performed to recover the variables $x_2, \cdots, x_n$ in the factors. Sparse Hensel lifting is an improvement upon the classical *multivariate Hensel lifting* (MHL) originally developed by Yun [58] and Wang [54]. Wang's multivariate Hensel lifting algorithm has been implemented in many computer algebra systems including Maple, Magma, Macsyma, Mathematica and Singular, and it is still widely used today.

To factor a multivariate polynomial $a \in \mathbb{Z}[x_1, x_2, \cdots, x_n]$, Wang's algorithm [54] first chooses integers $\alpha_2, \ldots, \alpha_n$ and factors the univariate image $a(x_1, \alpha_2, \ldots, \alpha_n)$ in $\mathbb{Z}[x_1]$. Then it recovers the multivariate factors from their images one variable at a time by solving *multivariate polynomial Diophantine equations* (MDPs) one variable at a time. For a detailed description of Wang's multivariate Hensel lifting we refer the reader to Chapter 6 of [19].

It is known that when factors are *sparse* and the evaluation points $\alpha_2, \ldots, \alpha_n$ are mostly non-zero, Wang's algorithm for solving MDPs can be exponential in the number of variables [43, 46]. In 1981, Zippel [61] gave the first polynomial-time randomized algorithm that takes advantage of sparsity. Another well-known sparse Hensel lifting (SHL) algorithm was developed by Kaltofen in 1985 [27].

In 2016, Monagan and Tuncer [43] contributed a new sparse Hensel lifting algorithm which they call MTSHL. It outperforms the previous sparse Hensel lifting algorithms and it solves the MDPs that appear in Wang's algorithm in random polynomial time. Algorithm MTSHL was integrated into Maple 2019 for multivariate polynomial factorization [47].

In 2018, Monagan and Tuncer [45] gave another approach that does not solve MDPs. Instead, at each Hensel lifting step, their algorithm interpolates the factors from many bivariate images obtained from bivariate Hensel lifts (BHL). The classical bivariate Hensel lifting algorithm by Bernardin [4] costs $O(d_x^2 d_y^2)$ arithmetic operations in $\mathbb{Z}_p$ on input $a \in \mathbb{Z}_p[x, y]$, where $d_x = \deg(a, x)$ and $d_y = \deg(a, y)$. In 2022, Monagan and Paluck [48] developed a cubic BHL algorithm which costs $O(d_x^2 d_y + d_x d_y^2)$ arithmetic operations in $\mathbb{Z}_p$.

Based on [45], Chen and Monagan [8] developed a highly parallelizable algorithm in 2020 which they call CMSHL. Algorithm CMSHL no longer does any multivariate polynomial arithmetic, and it eliminated a long standing problem of expression swell. A worst case complexity analysis for both MTSHL and CMSHL is presented in [8].

However, the dominating cost of CMSHL in practice is often evaluating the input polynomial $a(x_1, \ldots, x_n)$ at many points since the polynomial $a(x_1, \ldots, x_n)$ is stored using a *sparse representation*. For this reason, we considered the *black box representation* of a instead to reduce the cost of evaluating a. The memory space needed to store a in its sparse representation is also saved. An example is factoring the determinant of a matrix A with multivariate polynomial entries. Usually the factors of $a = \det A \in \mathbb{Z}[x_1, \ldots, x_n]$ have a lot fewer terms than a. Our goal is to compute the factors of a in their sparse representation.

In 1990, Kaltofen and Trager [31] contributed the first black box factorization algorithm for multivariate polynomials with coefficients in a field. Their algorithm first computes the black boxes of the factors, then the sparse representation of the factors can be recovered using *sparse polynomial interpolation*. Early references for sparse polynomial interpolation include [2, 32, 62]. For a recent bibliography we refer the reader to Roche [50]. In 1994, a simpler algorithm for factoring polynomials in $\mathbb{Z}[x_1, \cdots, x_n]$ was presented by Rubinfeld and Zippel [51]. Instead of using bivariate transformations to compute black boxes of the

factors in [31], Rubinfeld and Zippel's algorithm uses simple integer evaluations for each variable $x_2, \cdots, x_n$ and factors many univariate polynomials in $\mathbb{Z}[x_1]$.

Another implicit polynomial representation called the *straight-line program* (SLP) was considered prior to the black box representation. Several articles on factorization and computing the greatest common divisors with polynomials given by straight-line programs were published by Kaltofen in the late 1980's [29, 28, 30]. However, the black box representation is superior to the straight-line program model in many ways [31]. For example, in the SLP representation one must have access to the code. We have chosen to use the black box representation.

In 2022, Chen and Monagan [9] contributed a new black box factorization algorithm that computes the factors in the sparse representation directly by a modified CMSHL algorithm. The algorithm in [9] works if a is monic in x_1 and square-free. It outperforms Rubinfeld and Zippel's method as it requires fewer probes to the black box [10]. The non-monic, non-square-free and non-primitive cases have also been worked out and the complete factorization algorithm is presented in [10] with various timing benchmarks. This new algorithm is called CMBBSHL. A complexity analysis with failure probabilities for algorithm CMBBSHL is also given in [10].

If the polynomial a is *dense* then alternative methods for factoring should be considered. We cite the work of Lecerf [35] who factors a in $\mathbb{Q}[[x_2, \ldots, x_n]][x_1]$ and uses fast arithmetic for power series in $\mathbb{Q}[[x_2, \ldots, x_n]]$.

Another recent algorithm for sparse polynomial factorization that does not use Hensel lifting at all is described in a paper by Huang and Gao [23] in 2023. The main idea of their algorithm is that a multivariate polynomial $a(x_1, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_n]$ is reduced to a univariate polynomial with a substitution $x_i = p_i y^{s_i}$ for $i = 1, \cdots, n$, where s_i is selected at random from $[0, \frac{9}{2}T(T-1)]$. The integer $T = \sum_{\rho=1}^{r} \#f_\rho$ is the sum of the number of terms in the irreducible factors f_ρ. The distinct primes $p_1, \cdots, p_n$ are selected at random from $[N+1, 2N]$ for some large number N. Since the terms in the irreducible factors of $a(x_1 y^{s_i}, \cdots, x_n y^{s_n})$ are *separated* with respect to y with a probability $\geq 1 - \epsilon$ (i.e. each coefficient of $f_\rho(x_1 y^{s_i}, \cdots, x_n y^{s_n})$ in y contains only one monomial in $f_\rho(x_1, \cdots, x_n)$), the monomials can be recovered by trial divisions using the p_i's. Therefore, the problem of multivariate polynomial factorization is reduced to one univariate factorization. However, when T is large, a large degree univariate polynomial is created, and factoring it becomes expensive. Thus, their algorithm is not efficient for factoring polynomials with many terms.

1.2 Contributions

This thesis is based on the following three published papers co-authored with my PhD supervisor Prof. Michael Monagan.

1. T. Chen and M. Monagan. The complexity and parallel implementation of two sparse multivariate Hensel lifting algorithms for polynomial factorization. In Proceedings of CASC 2020, LNCS **12291**, 150–169. Springer (2020) [8]

2. T. Chen and M. Monagan. Factoring multivariate polynomials represented by black boxes: A Maple + C Implementation. *Math. Comput. Sci.* **16**,18 (2022) [9]

3. T. Chen and M. Monagan. A new black box factorization algorithm - the non-monic case. In Proceedings of ISSAC 2023, pp. 173–181. ACM (2023) [10]

The first paper improves upon Monagan and Tuncer's sparse Hensel lifting algorithm MTSHL [45]. I improved their algorithm to eliminate an expression swell, so it no longer does any multivariate polynomial arithmetic. It is also highly parallelizable. We call this algorithm CMSHL. I did a worst case complexity analysis for both MTSHL and CMSHL, presented in Section 4.4 of this thesis. In Chapter 4, Sections 4.3 and 4.4 are my new contributions.

The second paper presents a new black box factorization from a modified CMSHL algorithm (Approach II, as indicated in Figure 1.4). It works for $a(x_1, \cdots, x_n)$ monic in x_1 and square-free only. I designed the algorithm at Prof. Monagan's suggestion to use a black box representation. I implemented the algorithm with a Maple + C hybrid implementation. Two sets of timing benchmarks are presented. The first benchmark computes the determinant of symmetric Toeplitz matrices. Let T_n be an $n \times n$ symmetric Toeplitz matrix. I believe that I am the first to compute the factors of $\det(T_{16})$ with factors expressed in the sparse representation. I can factor $\det(T_{16})$ in 4877 seconds. Timings are presented in Section 5.3.2. In Chapter 5, Section 5.3 is my new contribution.

The third publication which was presented at ISSAC 2023, completes the black box factorization problem $\mathcal{P}2$. I call my new algorithm CMBBSHL. I designed and implemented the algorithm to handle all cases of input polynomials, i.e. non-monic, non-square-free and non-primitive cases. For the non-primitive case, I also implemented the code for recursively computing the content. After the conference in July 2023, I finished the benchmarks for content computation (Tables 6.3 – 6.7). A variety of timing benchmarks are presented in Section 6.3. I further considered the case for large integer coefficients and coded that in Maple. A timing benchmark for large integer coefficients is presented in Section 6.5. All of Chapter 6 is my new contribution.

I further improved the timings of computing the determinants of Toeplitz matrices $\det(T_n)$ for $n = 12, \cdots, 16$ by integrating a Maple implementation of the fast Vandermonde solver of Kaltofen and Yagati [32]. The new timing benchmarks are in Section 5.3 (see Tables 5.3 and 5.4).

1.3 Thesis outline

This thesis is organized as follows. Chapter 1 gives an introduction. Chapter 2 gives a code demo for my new black box factorization algorithm CMBBSHL. Chapter 3 describes essential tools (both mathematical and computational) used for designing both sparse Hensel lifting algorithms CMSHL and CMBBSHL. Chapter 4 presents details of algorithm CMSHL [8] which is used for factoring a polynomial in its sparse representation. Chapter 5 describes my new black box factorization algorithm CMBBSHL for the monic and square-free case only [9]. Chapter 6 presents the complete black box factorization algorithm CMBBSHL. It includes the non-monic, non-square-free, non-primitive and large integer cases with a variety of timing benchmarks. The chapter ends with a detailed complexity analysis with failure probabilities. Chapter 7 presents some implementation details of CMBBSHL. Chapter 8 is a summary of this thesis and a description of a few open problems.

1.4 Preliminary concepts and notations

We denote $\mathbb{Z}$ as the set of integers, $\mathbb{Z}^+$ as the set of positive integers, $\mathbb{N}$ the set of non-negative integers, and $\mathbb{Q}$ the set of rational numbers. Let $\overline{m}$ be the set of all integers which have the same remainder as m when divided by p, where p is a prime. The set $\{\overline{0}, \overline{1}, \cdots, \overline{p-1}\}$ is a field and it is denoted by $\mathbb{Z}_p$.

Let R be a commutative ring. Let $\mathrm{R}[x_1, \cdots, x_n]$ denote the set of all multivariate polynomials in the indeterminates $x_1, \cdots, x_n$ with coefficients lying in the ring R. For our problem of interest, $\mathrm{R} = \mathbb{Z}$, which is a unique factorization domain (UFD). If R is a UFD, then $\mathrm{R}[x_1, \cdots, x_n]$ is also a UFD (Theorem 2.7 in [19]).

We can also view the multivariate polynomial domain $\mathrm{R}[x_1, \cdots, x_n]$ as a univariate polynomial domain $(\mathrm{R}[x_2, \cdots, x_n])[x_1]$, for example, since $\mathrm{R}[x_1, \cdots, x_n]$ and $(\mathrm{R}[x_2, \cdots, x_n])[x_1]$ are *isomorphic*. Here, x_1 is a chosen *main variable* and the coefficients are in $\mathrm{R}[x_2, \cdots, x_n]$.

Definition 1.4.1. Let $a \in \mathrm{R}[x_1, \cdots, x_n]$. The *total degree* of a multivariate monomial $x_1^{e_1} \cdots x_n^{e_n}$ is $\sum_{j=1}^{n} e_j$, where $e_j \in \mathbb{N}$ [18]. The *total degree* of a, denoted as $\deg(a)$, is the maximal total degree of its monomials.

Definition 1.4.2. The *individual degrees* of $a \in \mathrm{R}[x_1, \cdots, x_n]$, denoted as $\deg(a, x_j)$, are the degrees of a in each variable x_j for $1 \leq j \leq n$.

Definition 1.4.3. Suppose that the terms in a non-zero multivariate polynomial $a \in \mathrm{R}[x_1, \cdots, x_n]$ have been arranged in *(pure or graded) lexicographically decreasing order* with $x_1 > x_2 > \cdots > x_n$. The first term is called the *leading term*, denoted as $\mathrm{LT}(a)$, and its coefficient is called the *leading coefficient*, denoted as $\mathrm{LC}(a)$.

Definition 1.4.4. Let $a \in \mathrm{R}[x_1, \cdots, x_n]$. If $\mathrm{LC}(a) = 1$, we say the polynomial a is *monic* (Section 2.6 of [19]). The leading coefficient of a in x_j is denoted as $\mathrm{LC}(a, x_j)$, a polynomial in $\mathrm{R}[x_2, \cdots, x_{j-1}, x_{j+1}, \cdots, x_n]$. If $\mathrm{LC}(a, x_j) = 1$, we say the polynomial a is *monic in* x_j.

We denote $\#a$ as the number of non-zero terms of $a \in \mathrm{R}[x_1, \cdots, x_n]$.

Note: For the context of the monic case of bivariate Hensel lifting, CMSHL and CMBBSHL, we mean a is *monic in* x_1.

Example 1. For example, with $w > x > y > z$,

$$\begin{aligned} a &= 11w^3xz - 49x^3yz - 29x^2y^2 + 95z^4 - 8w^3 - 61 \in \mathbb{Z}[w, x, y, z] \\ &= (95x)z^4 + (11w^3x - 49x^3y)z - 29x^2y^2 - 8w^3 - 61 \in \mathbb{Z}[w, x, y][z] \end{aligned}$$

Here, $\#a = 5$, $\deg(a) = 5$, $\mathrm{LC}(a) = 11$, $\deg(a, z) = 4$ and $\mathrm{LC}(a, z) = 95x$.

Definition 1.4.5. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ be a non-zero polynomial. Suppose the terms of a have been arranged in *(pure or graded) lexicographically decreasing order* with $x_1 > x_2 > \cdots > x_n$. We define the *sign* of a as

$$\mathrm{sign}(a) = \begin{cases} 1, & \text{if } \mathrm{LC}(a) \geq 0, \\ -1, & \text{if } \mathrm{LC}(a) < 0. \end{cases} \tag{1.1}$$

Definition 1.4.6. (Definition 2.18 in [19]) Let $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$. Let $d_1 = \deg(a, x_1)$ and $a = \sum_{i=1}^{d_1} a_i x_1^i$ with $a_i \in \mathbb{Z}[x_2, \cdots, x_n]$. The *content* of a, denoted as $\mathrm{Cont}(a)$, is the unique GCD of all coefficients of a s.t. $\mathrm{sign}(\mathrm{Cont}(a)) = 1$, i.e. $\mathrm{Cont}(a) = \gcd(a_0, a_1, \cdots, a_{d_1})$.

Note: For practical purpose of our algorithm design, the word 'content' is referred as a *generic content*, defined below.

Definition 1.4.7. Let $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$. The *generic content* $\mathrm{cont}(a)$ is defined as

$$\mathrm{cont}(a) = \mathrm{sign}(a) \cdot \mathrm{Cont}(a). \tag{1.2}$$

Definition 1.4.8. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$. The *integer content* of a is the unique non-negative GCD of all integer coefficients of a, denoted as $\mathrm{iCont}(a)$. Similar to the generic content, we also have the *generic integer content* defined as

$$\mathrm{icont}(a) = \mathrm{sign}(a) \cdot \mathrm{iCont}(a). \tag{1.3}$$

Definition 1.4.9. A non-zero polynomial $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$ is *primitive* if $\mathrm{Cont}(a) = 1$ and $\mathrm{sign}(a) = 1$. The *primitive part* of a, denoted as $\mathrm{pp}(a)$, is defined as

$$\mathrm{pp}(a) = \frac{a}{\mathrm{sign}(a) \cdot \mathrm{Cont}(a)} = \frac{a}{\mathrm{cont}(a)}. \tag{1.4}$$

Thus, any non-zero $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$ can be uniquely represented in the form

$$a = \mathrm{sign}(a) \cdot \mathrm{Cont}(a) \cdot \mathrm{pp}(a) = \mathrm{cont}(a) \cdot \mathrm{pp}(a). \tag{1.5}$$

Example 2. Let $a = (-3x_2 - 3x_3 - 3)x_1 - 3x_2^2 - 6x_2x_3 - 3x_3^2 - 6x_2 - 6x_3 - 3 \in \mathbb{Z}[x_2, x_3][x_1]$. Its irreducible factorization over $\mathbb{Z}$ is:

$$a = -3(x_2 + x_3 + 1)(x_1 + x_2 + x_3 + 1).$$

In this case, $\text{sign}(a) = -1$, $\text{Cont}(a) = 3(x_2 + x_3 + 1)$ (a is not primitive), $\text{cont}(a) = -3(x_2 + x_3 + 1)$, $\text{pp}(a) = x_1 + x_2 + x_3 + 1$, $\text{icont}(a) = -3$ and $\text{iCont}(a) = 3$.

1.5 Sparse polynomials and sparse representation

Various definitions for sparse polynomials exist in the literature. We shall use the following definition by Monagan [41] to distinguish a sparse polynomial from a dense polynomial.

Definition 1.5.1. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ and let $d = \deg(a)$ be the total degree of a. The maximum possible number of terms in a is $T = \binom{n+d}{d}$. We say a is *sparse* if $\#a \leq \sqrt{T}$.

Example 3. $a = 11x_1^5 + 4x_1^3x_2x_4 - 9x_1^3x_4 + 7x_1x_2^2x_3 + 20x_2^3x_3x_4$ is sparse. In this case, $n = 4$, $d = 5$, $T = \binom{n+d}{d} = 126$. $\sqrt{T} \approx 11.225$ and $\#a = 5 < \sqrt{T}$.

One might be inclined to think that sparse polynomials also have sparse irreducible factors. However, this is not always the case as shown in the following example [27].

Example 4. Let $a = x^{401}y^{23} - x^{401} - y^{23} + 1 \in \mathbb{Z}[x, y]$. Factoring a gives

$$a = (y-1)\underbrace{(y^{22} + y^{21} + \cdots + y + 1)}_{23 \text{ terms}}(x-1)\underbrace{(x^{400} + x^{399} + \cdots + x + 1)}_{401 \text{ terms}}.$$

In this case, $n = 2$, $d = 424$, $T = \binom{n+d}{d} = 90525$ and $\sqrt{T} \approx 300.87$. $\#a = 4 < 300.87$ so a is sparse by Definition 1.5.1. However, one of the factors of a has 401 terms and it is not sparse as $401 > \sqrt{T}$.

The current best known bound for the number of terms of the factors of a polynomial is give by Bhargava et al [5]. They proved that for $a \in \mathbb{F}[x_1, \cdots, x_n]$ where $\mathbb{F}$ is a field, with individual degrees of its variables bounded by $d_{\max}$, i.e. $\deg(a, x_j) \leq d_{\max}$ for all $1 \leq j \leq n$, the number of terms of each factor of f is bounded by $(\#a)^{O(d_{\max}^2 \log n)}$.

However, in practice and in theory, the number of terms of each irreducible factor of a is almost always less than $\#a$.

On the other hand, from a computer science standpoint, a sparse or dense polynomial can also be classified by its data structure or *representation* [50]. A *dense representation* of a polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$ stores its zero coefficients. For example, the polynomial $a = x^3 + y^3 - 2xy \in \mathbb{Z}[x, y]$ can be stored as an array of arrays

$$[[0, 0, 0, 1], [0, -2, 0, 0], [0, 0, 0, 0], [1, 0, 0, 0]].$$

The sub-arrays represent the coefficients of a in x (i.e. a polynomial in $\mathbb{Z}[y]$ represented by coefficients of y^0, y^1, y^2, y^3).

There is also a *recursive dense representation* [17]. For example, in Maple's RECDEN data structure [22], the polynomial $x^3 + y^3 - 2xy$ is stored as

$$[[0, 0, 0, 1], [0, -2], 0, [1]].$$

For a sparse representation, we employ the following definition (Section 16.6 of [18]):

Definition 1.5.2. The *sparse representation* of a polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$ consists of a list of coefficients $c_k \neq 0$, $c_k \in \mathbb{Z}$ and distinct exponent vectors $\vec{e_k} = (e_{k_1}, \cdots, e_{k_n}) \in \mathbb{N}^n$ s.t.

$$a = \sum_{k=1}^{\#a} c_k \cdot x_1^{e_{k_1}} \cdots x_n^{e_{k_n}}, \tag{1.6}$$

where $\#a$ is the number of non-zero terms of a.

Definition 1.5.3. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ be represented as in (1.6). The *max-norm* (also known as the *height*) of a, denoted as $\|a\|_\infty$ is defined as

$$\|a\|_\infty = \max_{k=1}^{\#a} |c_k|. \tag{1.7}$$

It is the largest absolute value of the coefficients of a. E.g. $\|2x_1 - 3\|_\infty = 3$.

Example 5. Maple's POLY data structure [42] is a sparse representation. Figure 1.1 illustrates how Maple stores the polynomial $a = 5x_4^9 - 2x_2^3x_3^5 + x_1^4$. The exponents are stored as 64-bit integers with their total degrees. Each coefficient is stored next to the exponent.

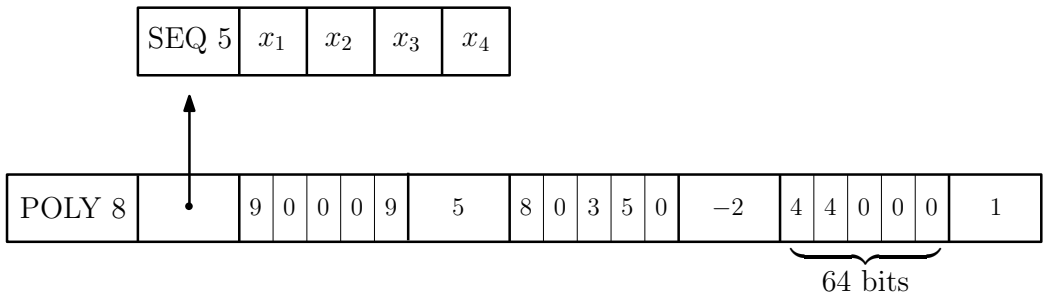

Figure 1.1: Maple's POLY data structure for $a = 5x_4^9 - 2x_2^3x_3^5 + x_1^4$.

A monomial $x_1^i x_2^j x_3^k x_4^l$ is encoded in 64 bits as the integer

$$(i + k + j + l) \cdot 2^{48} + i \cdot 2^{36} + j \cdot 2^{24} + k \cdot 2^{12} + l$$

where we use $\lfloor 64/5 \rfloor = 12$ bits for i, j, k, l and 16 bits for $i+j+k+l$. Monomials can then be compared in the graded lexicographic order with $x_1 > x_2 > x_3 > x_4$ using a 64-bit integer comparison and multiplied using a 64-bit integer addition provided no overflow occurs.

This representation has a limit for the number of variables stored. The maximum number of variables can be stored using POLY data structure is 31.

1.6 The black box representation of a polynomial

The sparse representation as defined in (1.6) is a natural, readable and *explicit* representation. On the other hand, the black box representation defined in Definition 1.6.1 is *implicit*.

Definition 1.6.1. Let $a \in \mathrm{R}[x_1, \cdots, x_n]$ where R is an integral domain, e.g. $\mathrm{R} = \mathbb{Z}$. The *black box representation* of a is a computer program $\mathbf{BB} : \mathrm{R}^n \to \mathrm{R}$ that on input $\boldsymbol{\alpha} \in \mathrm{R}^n$ computes $a(\boldsymbol{\alpha})$, i.e. $\mathbf{BB}(\boldsymbol{\alpha}) = a(\boldsymbol{\alpha})$ (see Figure 1.2).

Figure 1.2: The black box representation of $a \in \mathrm{R}[x_1, ..., x_n]$.

We cannot see inside the black box **BB**. The only thing we can do is to call the black box at a given point $\boldsymbol{\alpha} \in \mathrm{R}^n$ and obtain its evaluation $\mathbf{BB}(\boldsymbol{\alpha})$.

Definition 1.6.2. We define one function call to the black box **BB** as one **probe** to **BB**.

Since algorithm CMBBSHL is modular, we use a modular black box for $a \in \mathbb{Z}[x_1, \cdots, x_n]$, defined below.

Definition 1.6.3. A *modular black box representation* of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ is a computer program $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ that on input $\boldsymbol{\alpha} \in \mathbb{Z}^n$ and a prime p outputs $\mathbf{B}(\boldsymbol{\alpha}, p) = a(\boldsymbol{\alpha}) \bmod p$ (see Figure 1.3).

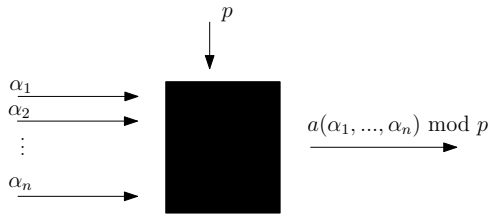

Figure 1.3: A modular black box representation of $a \in \mathbb{Z}[x_1, ..., x_n]$.

1.6.1 The Schwartz-Zippel lemma

Given a black box representation **BB** for a polynomial $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$, what information can we get from it? Can we determine whether $f = 0$? Or $f = c$, a constant function? In order to answer these questions, we first need the **Schwartz-Zippel lemma** (also known as the DeMillo-Lipton-Schwartz-Zippel lemma [52, 60, 15]).

Lemma 1.6.4. (Schwartz-Zippel lemma) *Let* R *be an integral domain and let* $S \subseteq$ R *be finite. Let* $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ *be a non-zero polynomial with total degree* d*. Then the number of roots of* f *in* S^n *is at most* $d|S|^{n-1}$*. Hence if* $\boldsymbol{\beta}$ *is chosen at random from* S^n*, then* $\Pr[f(\boldsymbol{\beta}) = 0] \leq \frac{d}{|S|}$.

The Schwartz-Zippel lemma is often used for probabilistically determining whether a given multivariate polynomial is identically zero by evaluating the polynomial at a single point. Our sparse Hensel lifting algorithms are modular, i.e. arithmetic operations are performed in $\mathbb{Z}_p$ during sparse Hensel lifting steps. If a large prime p is used, and the total degree d of $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$ ($f \neq 0$) is much less than p, i.e. $d \ll p$, then the probability of evaluating f to be zero is $\ll 1$.

A simple way to determine whether $f = 0$ is to evaluate f at a random point $\boldsymbol{\alpha} \in {\mathbb{Z}_p}^n$ and see if $\mathbf{BB}(\boldsymbol{\alpha}) = 0$. If $\mathbf{BB}(\boldsymbol{\alpha}) \neq 0$, then $f \neq 0$. If $\mathbf{BB}(\boldsymbol{\alpha}) = 0$, then $f = 0$ with high probability (w.h.p.). The failure probability is bounded by d/p, by Lemma 1.6.4.

To determine whether f is a constant function, we first probe the black box **BB** at $\boldsymbol{\beta}_1 = (0, 0, \cdots, 0)$ and let $c_1 = \mathbf{BB}(\boldsymbol{\beta}_1)$. Then another point $\boldsymbol{\beta} \in {\mathbb{Z}_p}^n$ is chosen at random to test if $\mathbf{BB}(\boldsymbol{\beta}) - c_1 = 0$. If not, then f is not a constant function. If yes, f is a constant w.h.p. and the failure probability is mostly d/p, again by Lemma 1.6.4.

In fact, if a polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$ is represented by a black box **BB** (or a modular black box **B**), the sparse representation of a can be computed in random polynomial time by sparse polynomial interpolation (see for example, [2, 32, 62]).

Without interpolating the polynomial a, we can also determine some other information. For example, the total degree of a (see Section 1.6.2 for details) and the individual degrees of a (Section 3.1) can be computed w.h.p.

1.6.2 Computing the total degree of a polynomial

Given a modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ for $a \in \mathbb{Z}[x_1, \cdots, x_n]$, we can compute the total degree of a, $\deg(a)$, with high probability (see Algorithm 3). The method presented in this section is based on Prof. Monagan's lecture notes in Computer Algebra [40], which is different from the algorithm by Kaltofen and Trager [31].

In order to compute $\deg(a)$, the multivariate polynomial a is first transformed to a univariate polynomial by the following formula

$$g(z) = a(\beta_1 z, \beta_2 z, \cdots, \beta_n z) \in \mathbb{Z}[z] \tag{1.8}$$

Algorithm 1 Construct $\mathbf{U}_g : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$ for $g(z) \in \mathbb{Z}[z]$.

Input: A black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ representing $a \in \mathbb{Z}[x_1, \cdots, x_n]$,
a large prime p, $\beta_1, \cdots, \beta_n \in \mathbb{Z}_p$.
Output: A black box $\mathbf{U}_g : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{U}_g(\gamma, p) = g(\gamma) \bmod p$.

$\mathbf{U}_g$ = procedure(γ, p) // $\gamma \in \mathbb{Z}$
  $\boldsymbol{\beta}_\gamma \leftarrow (\beta_1\gamma, \cdots, \beta_n\gamma)$.
  return $\mathbf{B}(\boldsymbol{\beta}_\gamma, p)$ // $\mathbf{B}(\boldsymbol{\beta}_\gamma, p) = a(\beta_1\gamma, \cdots, \beta_n\gamma) \bmod p = g(\gamma) \bmod p$.
end procedure
return $\mathbf{U}_g$

Algorithm 2 UnivDeg: Compute the degree of $f(x) \in \mathbb{Z}[x]$ w.h.p.

Input: A black box $\mathbf{U} : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$ representing $f(x) \in \mathbb{Z}[x]$, a large prime p.
Output: $\deg(f)$ with high probability.

$g_{-1} \leftarrow 0$; $k \leftarrow 0$; $m \leftarrow 1$.
while true **do**
  Pick $\alpha_k \in \mathbb{Z}_p$ at random s.t. $m(\alpha_k) \neq 0$.
  $y_k \leftarrow \mathbf{U}(\alpha_k, p)$. //$y_k = f(\alpha_k) \bmod p$
  $v_k \leftarrow (y_k - g_{k-1}(\alpha_k))/m(\alpha_k)$.
  if $v_k = 0$ **then return** $k - 1$ **end if**
  $g_k \leftarrow g_{k-1} + v_k \cdot m$.
  $m \leftarrow m \cdot (x - \alpha_k)$.
  $k \leftarrow k + 1$.
end while

Algorithm 3 TotalDeg: Compute the total degree of $a(x) \in \mathbb{Z}[x_1, \cdots, x_n]$ w.h.p.

Input: A modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ representing $a(x) \in \mathbb{Z}[x_1, \cdots, x_n]$.
Output: $\deg(a)$ with high probability.

Pick a large prime p.
Pick $\beta_1, \cdots, \beta_n \in \mathbb{Z}_p$ at random.
Construct a black box $\mathbf{U}_g : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$ for $g(z) = a(\beta_1 z, \cdots, \beta_n z)$ by Algorithm 1.
return UnivDeg($\mathbf{U}_g$, p)

where $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_n) \in {\mathbb{Z}_p}^n$ is chosen at random. We construct a modular black box $\mathbf{U}_g : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$ for $g(z) \in \mathbb{Z}[x]$ by Algorithm 1.

By computing the degree of $g(z) \in \mathbb{Z}[x]$ w.h.p. (Algorithm 2), we obtain $d = \deg(a)$ w.h.p. (Algorithm 3). The failure probabilities of Algorithm 2 and Algorithm 3 are given by Proposition 1.6.5 and Theorem 1.6.7 respectively.

Algorithm 2 (`UnivDeg`) computes the degree of a univariate polynomial $f(x) \in \mathbb{Z}[x]$ represented by a modular black box $\mathbf{U} : \mathbb{Z} \times \{p\} \to \mathbb{Z}_p$. Let $\alpha_0, \alpha_1, \cdots, \alpha_d$ be distinct points

in $\mathbb{Z}_p$ and suppose $d < p$. Let $f(x) \in \mathbb{Z}[x]$ be written in the Newton form as

$$\begin{aligned} f(x) = v_0 + v_1(x-\alpha_0) + v_2(x-\alpha_0)(x-\alpha_1) + \cdots + v_d(x-\alpha_0)\cdots(x-\alpha_{d-1}) \\ + \underbrace{v_{d+1}}_{=0}(x-\alpha_0)\cdots(x-\alpha_d) + \cdots \end{aligned}$$

where $v_k \in \mathbb{Z}$ for $k = 0, 1, \cdots$, and $d = \deg(f)$. This renders unique v_k's with $v_d \neq 0$ and $v_k = 0$ for $k \geq d+1$. Algorithm 2 computes v_k for $k = 0, 1, \cdots$, until the first $v_k = 0$.

Proposition 1.6.5. *Let $f(x) \in \mathbb{Z}[x]$ and $d = \deg(f)$. Let p be a large prime and suppose $d < p$. Assume $\deg(f \mod p) = d$. Let d_U be the output from Algorithm 2 (*`UnivDeg`*). Then,*

$$\Pr[d_U < d] \leq \frac{d^2}{p-d+1}. \tag{1.9}$$

Proof. By assumption, $\deg(f) = \deg(f \bmod p)$. Thus, $v_d \neq 0 \bmod p$. Thus, Algorithm 2 returns $d_U < d$ if $v_k = 0$ for some $k \in \{0, \cdots, d-1\}$.

$$v_k = 0 \iff \underbrace{f(\alpha_k)}_{y_k} - g_{k-1}(\alpha_k) = 0.$$

Let $h(x) = f(x) - g_{k-1}(x)$. Since $\deg(f) = d$ and $\deg(g_{k-1}) = k-1$, $\deg(h) = d$. Since there are $p-k$ choices for $\alpha_k \notin \{\alpha_0, \cdots, \alpha_{k-1}\}$, we have for $k \in \{0, \cdots, d-1\}$,

$$\Pr[v_k = 0] = \Pr[h(\alpha_k) = 0] \leq \frac{\deg(h)}{p-k} = \frac{d}{p-k}.$$

Thus,

$$\Pr[v_0 = 0 \text{ or } v_1 = 0 \text{ or } \cdots \text{ or } v_{d-1} = 0] \leq \frac{d}{p} + \frac{d}{p-1} + \cdots + \frac{d}{p-d+1} \leq \frac{d^2}{p-d+1}.$$

□

W.l.o.g, let p be a 63-bit prime, i.e. $p \in (2^{62}, 2^{63})$. Let $\mathbb{P}_{63} = \{\text{all 63-bit primes}\}$. The number of 63-bit primes, $|\mathbb{P}_{63}|$, can be estimated using the **prime number theorem**, i.e.

$$\pi(N) \sim \frac{N}{\log(N)}, \tag{1.10}$$

where $\pi(N)$ is the *prime-counting function* that counts the number of primes less than or equal to $N \in \mathbb{Z}^+$ and $\log(N)$ is the natural logarithm of N.

The number of 63-bit primes is approximately

$$|\mathbb{P}_{63}| = \pi(2^{63}) - \pi(2^{62}) \sim \frac{2^{63}}{\log(2^{63})} - \frac{2^{62}}{\log(2^{62})} \approx 1.039 \times 10^{17}. \tag{1.11}$$

Choosing a prime at random from $\mathbb{P}_{63}$ is non-trivial. We shall discuss this problem in Chapter 8. For now, we assume this can be done computationally.

Proposition 1.6.6. *Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$. Let $g(z) = a(\beta_1 z, \beta_2, z, \cdots, \beta_n z) \in \mathbb{Z}[z]$ where $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_n) \in {\mathbb{Z}_p}^n$ is chosen at random. Let $a_z = a(x_1 z, \cdots, x_n z) \in \mathbb{Z}[x_1, \cdots, x_n][z]$ and let H_s be the smallest absolute value of the coefficients of $\mathrm{LC}(a_z)$. Let $d = \deg(a)$ and $p \in \mathbb{P}_{63}$ be chosen at random. Then,*

$$\Pr[\deg(g \bmod p) < \deg(a)] \leq \frac{d}{p} + \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}. \tag{1.12}$$

Proof. Let $d_g = \deg(g \bmod p)$. We have

$$d_g < d \iff \mathrm{LC}(a_z)(\boldsymbol{\beta}) = 0 \bmod p.$$

This happens if $p | \mathrm{LC}(a_z)$ or if $p \nmid \mathrm{LC}(a_z)$, $\mathrm{LC}(a_z)(\boldsymbol{\beta}) = 0 \bmod p$.
Since H_s has at most $\lfloor \log_2(H_s)/62 \rfloor$ prime divisors from $\mathbb{P}_{63}$,

$$\Pr[p | \mathrm{LC}(a_z)] \leq \Pr[p | H_s] \leq \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}.$$

Thus,

$$\begin{aligned} \Pr[d_g < d] &= \Pr[\mathrm{LC}(a_z)(\boldsymbol{\beta}) = 0 \bmod p \mid p \nmid \mathrm{LC}(a_z)] + \Pr[p | \mathrm{LC}(a_z)] \\ &\leq \frac{d}{p} + \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}. \end{aligned}$$

The first term above is obtained from the Schwartz-Zippel lemma (Lemma 1.6.4). □

Example 6. Let $a = 7x_1x_3^4 - 13x_2^3x_3^2 + 2x_1x_2^2 + 3x_1^2 \in \mathbb{Z}[x_1, \cdots, x_3]$. Let $p = 2^{62} + 135$. Then, $a_z = (7x_1x_3^4 - 13x_2^3x_3^2)z^5 + 2x_1x_2^2z^3 + (3x_1^2)z^2$. $\mathrm{LC}(a_z) = 7x_1x_3^4 - 13x_2^3x_3^2$ and $H_s = 7$. We see that $d = 5$ and

$$d_g < d \iff 7\beta_1\beta_3^4 - 13\beta_2^3\beta_3^2 = 0 \bmod p.$$

Since $\Pr[p | \mathrm{LC}(a_z)] = 0$ and $\Pr[\mathrm{LC}(a_z)(\boldsymbol{\beta}) = 0 \bmod p \mid p \nmid \mathrm{LC}(a_z)] \leq d/p$,

$$\Pr[d_g < d] \leq \frac{d}{p} = \frac{5}{p} \approx 1.084 \times 10^{-18}.$$

Theorem 1.6.7. *Let p be a prime chosen randomly from $\mathbb{P}_{63}$. Let $a_z = a(x_1 z, \cdots, x_n z) \in \mathbb{Z}[x_1, \cdots, x_n][z]$. Let H_s be the smallest absolute value of the coefficients of a_z. Let $d =$*

$\deg(a)$ *be the total degree of* a *and let* d_T *be the output from Algorithm 3. Then,*

$$\Pr[d_T < d] \leq \frac{d}{p} + \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|} + \frac{d^2}{p-d+1}. \tag{1.13}$$

Proof. Let $d_g = \deg(g \bmod p)$ where $g(z) = a(\beta_1 z, \beta_2, z, \cdots, \beta_n z)$. We have

$$\begin{aligned}
\Pr[d_T < d] &= \Pr[d_T < d | d_g < d]\Pr[d_g < d] + \Pr[d_T < d | d_g = d]\Pr[d_g = d] \\
&\leq 1 \cdot \underbrace{\left(\frac{d}{p} + \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}\right)}_{\text{Proposition 1.6.6}} + \underbrace{\left(\frac{d^2}{p-d+1}\right)}_{\text{Proposition 1.6.5}} \cdot \underbrace{\Pr[d_g = d]}_{\leq 1} \\
&\leq \frac{d}{p} + \left\lfloor \frac{\log_2(H_s)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|} + \frac{d^2}{p-d+1}.
\end{aligned}$$

□

1.7 Factoring a polynomial represented by a black box

1.7.1 Problem description

Problem $\mathcal{P}2$: Given a polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$ represented by a *black box* $\mathbf{BB} : \mathbb{Z}^n \to \mathbb{Z}$ or a *modular black box* $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$, compute its irreducible factors in $\mathbb{Z}[x_1, \cdots, x_n]$ in their *sparse representation* but do not factor the integer content.

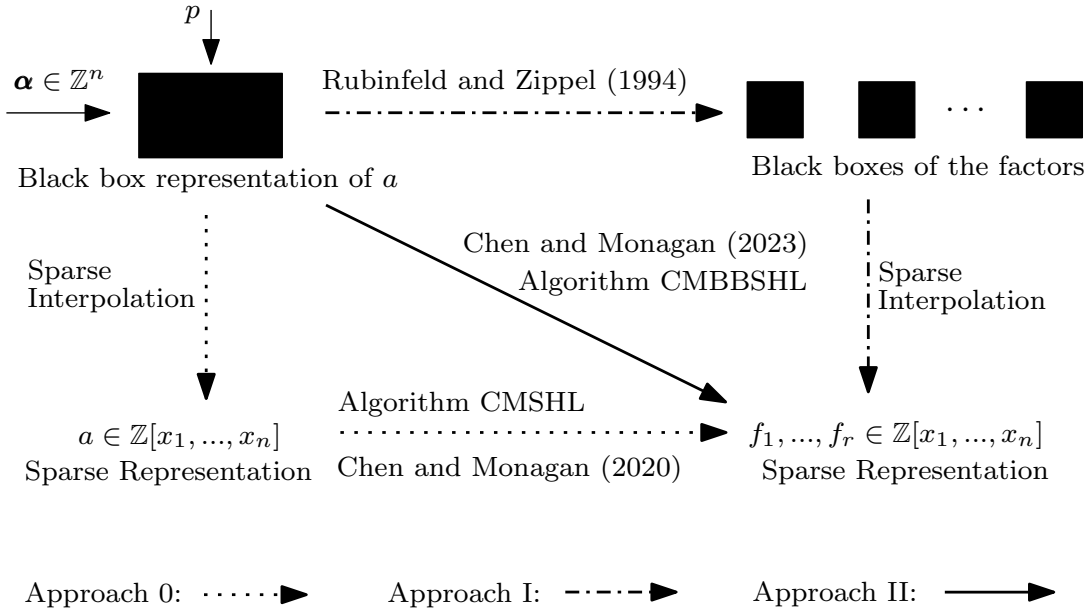

Figure 1.4: Factoring $a \in \mathbb{Z}[x_1, ..., x_n]$ represented by a black box.

Figure 1.4 shows three approaches for Problem $\mathcal{P}2$. Approach 0 first interpolates the sparse representation of a and then factors it using a sparse Hensel lifting algorithm, e.g. algorithm CMSHL [8] (described in Chapter 4). Approach I uses the black box representation **BB**. It first constructs black boxes of the factors and then applies sparse polynomial interpolation (e.g. [2, 62]) to obtain the sparse representation of the factors, e.g, Kaltofen and Trager's algorithm [31] and Rubinfeld and Zippel's algorithm [51]. Approach II (Algorithm CMBBSHL) uses the modular black box representation **B**. It computes the factors in the sparse representation directly by a modified CMSHL algorithm [9, 10]. Algorithm CMBBSHL is my new contribution and it is described in Chapter 5 and Chapter 6. Approach II is a modular algorithm and it is the most efficient of the three for two reasons. The first reason is that Algorithm CMBBSHL requires fewer probes to the black box than both Kaltofen and Trager's algorithm and Rubinfeld and Zippel's algorithm. The second and the key reason is that Algorithm CMBBSHL works mod p while both Kaltofen and Trager's algorithm [31] and Rubinfeld and Zippel's algorithm [51] work over $\mathbb{Q}$.

1.7.2 Example of a determinant computation

Consider the problem of computing the irreducible factors of the determinant of a symmetric Toeplitz matrix where

$$T_n = \begin{pmatrix} x_1 & x_2 & x_3 & \cdots & x_n \\ x_2 & x_1 & x_2 & & \\ x_3 & x_2 & x_1 & & \\ \vdots & & & \ddots & \vdots \\ x_n & & & \cdots & x_1 \end{pmatrix}. \tag{1.14}$$

For example, if $n = 4$, then

$$\det(T_4) = (x_1^2 - x_1x_2 - x_1x_4 - x_2^2 + 2x_2x_3 + x_2x_4 - x_3^2)(x_1^2 + x_1x_2 + x_1x_4 - x_2^2 - 2x_2x_3 + x_2x_4 - x_3^2)$$

which has two factors and each has 7 terms. If $\det(T_4)$ is expanded, it has 12 terms.

The determinant of T_n has two factors for all $n \geq 2$. If n is even, the number of terms is the same for both factors. The factors of $\det(T_n)$ are generally dense, according to Definition 1.5.1. For example, when $n = 10$, both factors of $\det(T_n)$ have 931 terms and a total degree of 5. Thus, $\binom{n+5}{5} = 3003$, and $\#f_i = 931 > \sqrt{3003} \approx 54.8$. It is worth noting that our algorithm works well for both sparse and dense polynomials.

Let $a = \det(T_n) \in \mathbb{Z}[x_1, \cdots, x_n]$. The modular black box representation **B** of a can be coded in Maple as a procedure:

```
B := proc( alpha::Array, p::prime )
    local n := numlems(alpha), i,j,Tn;
    Tn := Matrix(n,n);
```

```
    for i to n do
        for j to n do
            Tn[i,j] := alpha[abs(i-j)+1];
        od;
    od;
    Det(Tn) mod p;
end:
```

Overview of algorithm CMBBSHL applied to Toeplitz determinant

Algorithm CMBBSHL first chooses an evaluation point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ (see Section 3.2 for details), then $a(x_1, \boldsymbol{\alpha}) \bmod p \in \mathbb{Z}_p[x_1]$ is computed w.h.p. from $d_1 + 1$ ($d_1 = \deg(a, x_1)$, pre-computed w.h.p., see Section 3.1 for details) probes to the black box **B** and a Lagrange (or Newton) interpolation. This process is called a **univariate dense interpolation**. Then $a(x_1, \boldsymbol{\alpha}) \in \mathbb{Z}[x_1]$ is computed using Chinese remaindering with different primes to recover the coefficients in $\mathbb{Z}$. Next, $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}[x_1]$. For example, if $\boldsymbol{\alpha} = (3, 5, 4)$,

$$a(x_1, \boldsymbol{\alpha}) = x_1^4 - 93x_1^2 + 420x_1 - 416 = (x_1^2 - 7x_1 + 8)(x_1^2 + 7x_1 - 52).$$

If we choose $p = 101$, the initial input factors of algorithm CMBBSHL are $x_1^2 - 7x_1 + 8$ and $x_1^2 + 7x_1 + 49$. The first Hensel lift (a bivariate Hensel lift) recovers x_2 and we get

$$\begin{aligned} f_2 &= x_1^2 - x_1x_2 - x_2^2 - 4x_1 + 14x_2 - 25, \\ g_2 &= x_1^2 + x_1x_2 - x_2^2 + 4x_1 - 6x_2 - 25. \end{aligned}$$

The second Hensel lifting step recovers x_3, and we get

$$\begin{aligned} f_3 &= x_1^2 - x_1x_2 - x_2^2 + 2x_2x_3 - x_3^2 - 4x_1 + 4x_2, \\ g_3 &= x_1^2 + x_1x_2 - x_2^2 - 2x_2x_3 - x_3^2 + 4x_1 + 4x_2. \end{aligned}$$

And at the final Hensel lifting step, we recover x_4 and the true factors

$$\begin{aligned} f &= x_1^2 - x_1x_2 - x_1x_4 - x_2^2 + 2x_2x_3 + x_2x_4 - x_3^2, \\ g &= x_1^2 + x_1x_2 + x_1x_4 - x_2^2 - 2x_2x_3 + x_2x_4 - x_3^2. \end{aligned}$$

Algorithm CMBBSHL calls the black box **B** in each Hensel lifting step, but it does not compute the sparse representation of $a = \det(T_n)$. As n increases, $\#a$ becomes very large, but the factors have fewer terms. The memory space to store a in its sparse representation is saved. Table 1.1 shows the number of terms of $\det(T_n)$ and the number of terms in each factor of $\det(T_n)$ for n up to 16. Although we could compute the factors of $\det(T_{16})$, we

could not multiply them out in Maple since Maple ran out of memory on an Intel Xeon E5-2660 8 core CPU (64 GB RAM).

n	$\#\det(T_n)$	$\#f_i$
8	1628	$167, 167$
9	6090	$294, 153$
10	23797	$931, 931$
11	90296	$1730, 849$
12	350726	$5579, 5579$
13	1338076	$10611, 4983$
14	5165957	$34937, 34937$
15	19732508	$66684, 30458$
16	–	$221854, 221854$

Table 1.1: Number of terms of $\det(T_n)$ and its factors ($\#f_i$).

1.8 Randomized algorithms

Our sparse Hensel lifting algorithms CMSHL and CMBBSHL for multivariate polynomial factorization are *randomized*.

A *randomized algorithm* is an algorithm whose behavior depends not only on the input, but also by values that produced by a *random-number generator* [13]. The main advantage of using a randomized algorithm is that it is often more efficient than the best known *deterministic algorithm*. As shown in complexity analyses [8, 10], both algorithms CMSHL and CMBBSHL are *random polynomial time* algorithms (defined below).

Definition 1.8.1. A randomized algorithm (or probabilistic algorithm) that runs in expected polynomial time is called a *random polynomial time* algorithm.

There are three types of randomized algorithms: Las Vegas, Monte Carlo, and Atlantic City algorithms. A *Las Vegas algorithm* always produces the correct output or outputs a message of failure but it may get unlucky, and take a long time but with a low probability. A *Monte Carlo algorithm* is a random polynomial time algorithm that answers correctly at least 50% of the time. An *Atlantic City algorithm* is a random polynomial time algorithm that answers correctly at least 75% of the time [37].

Las Vegas algorithms are always correct, but only probably fast. Monte Carlo algorithms are always fast, but only probably correct. Atlantic City algorithms are both probably correct and probably fast [53].

Algorithm CMSHL is Las Vegas and CMBBSHL is Monte Carlo. In fact, algorithm CMBBSHL is Atlantic City, as the probability of returning an incorrect answer is very low (see Section 6.4). There has been active research on *deterministic* algorithms for factoring multivariate polynomials. The current best known is a quasi-polynomial deterministic algorithm by Bhargava et al [5].

Chapter 2

Implementation demonstration

My implementation of algorithm CMBBSHL is a hybrid of Maple and C code. The main program is in Maple and it consists of four major subroutines:

1. probes to the black box and bivariate dense interpolation,
2. evaluations of the partially recovered factors $f_\rho(x_1, \cdots, x_{j-1}, \alpha_j, \cdots, \alpha_n)$,
3. bivariate Hensel lifts,
4. Vandermonde solves.

All of the above subroutines are coded in C to speed up computations.

My contributions to the implementation of algorithm CMBBSHL include the following: the main program in Maple, which includes Maple to C interface functions to call all four major sub-routines coded externally in C, and the C code for bivariate dense interpolation and determinant computation of symmetric Toeplitz matrices using Bareiss' $\mathcal{O}(n^2)$ algorithm [1]. I also wrote Maple code for black box constructions. In particular, given a matrix A with multivariate polynomial entries, I wrote a Maple program to output a black box $\mathbf{B}$ which takes input $\boldsymbol{\alpha} \in \mathbb{Z}^n$, a prime p, and outputs $\det(A(\boldsymbol{\alpha})) \bmod p$.

For implementation details, see Chapter 7. For details of algorithm CMBBSHL, see Section 5.3 and Chapter 6. The complete code for algorithm CMBBSHL is on the website `http://www.cecm.sfu.ca/~mmonagan/code/CMBBSHL/`. Appendix A presents the code to run CMBBSHL.

In this chapter, we present two examples. The first example has only two factors, which is very simple, and it illustrates the basic idea of algorithm CMBBSHL. The second example is to compute the irreducible factors of a determinant of a large matrix and it is more complicated.

2.1 A simple two-factor example

Let us compute the factors of $a = x_1x_2 + x_1x_3 + x_2^2 + 2x_2x_3 + x_3^2 + x_1 + 2x_2 + 2x_3 + 1$ over $\mathbb{Z}$. The factorization is

$$a = (x_2 + x_3 + 1)(x_1 + x_2 + x_3 + 1) \in \mathbb{Z}[x_1, x_2, x_3].$$

If the main variable is chosen to be x_1, a has a content which is $x_2 + x_3 + 1$. If we choose the main variable to be x_2, a has no content but both factors are monic (in x_2). We first look at the latter case.

The input modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ is coded in Maple as a Maple procedure:

```
BBInput := proc( alpha::Array, p::prime ) global CNT; CNT++;
    alpha[1]*alpha[2]+alpha[1]*alpha[3]+alpha[2]^2+2*alpha[2]*alpha[3]
    +alpha[3]^2+alpha[1]+2*alpha[2]+2*alpha[3]+1 mod p;
end;
```

Other inputs for procedure `CMBBSHLcont`, the main program for algorithm CMBBSHL, are (see Appendix A for detailed descriptions):

```
X := [x2, x1, x3]: # Variables with a chosen ordering
alpha := Array(1..3, [2908,3830,2798]): # Evaluation point chosen at random
degA := [2, 1, 2]: # Pre-computed individual degrees
p := prevprime(2^62-1): # A chosen large prime number
Var_Perm := 1: # Yes, X is permuted
Cont_Flag := 1: # Yes, compute and factor the content recursively
sqfinterp := 0: # An option to use SquareFreeImage, turned off
MapleCode := [Maple,C,C,C]: # Use Maple or C code for each subroutine
LI := 0: # Not for large coefficients
```

Running `CMBBSHLcont` produces the following output:

```
> CNT := 0:
> ff := CMBBSHLcont( BBInput, X, alpha, degA, p, Var_Perm, Cont_Flag,
        sqfinterp, MapleCode, LI );
CMBBSHLcont: N = 3
1 prime(s) used to recover a(x2) = a(alpha1,x2,alpha3)
a(x2) = x2^2+9428*x2+18554571
N = 3, factors of a(x2) = [[x2+6629, 1], [x2+2799, 1]] in Z[x2]
CMBBSHL step 3: s = 2
fN = [x1+x2+x3+1, x2+x3+1]
Total time for CMBBSHLcont at N=3 (Computing F_pp) = 0.016000s
```

```
CMBBSHLcont: N = 2
Total time for CMBBSHLcont at N=2 (deg(a,x1)=0, MakeCont only) = 0.000000s
CMBBSHLcont: N = 1
Total time for CMBBSHLcont at N=1 (deg(a,x3)=0, MakeCont only) = 0.000000s
CMBBSHLcont: N = 0
1 prime(s) used to recover content
Total time for CMBBSHLcont at N=0 = 0.000000s
N = 0, integer content = 1
N = 1, content (in the variable x3) = 1
N = 2, content (in the variable x1) = 1
N = 3, f_pp = (x1+x2+x3+1)*(x2+x3+1), content (in the variable x2) = 1

                ff := (x1 + x2 + x3 + 1) (x2 + x3 + 1)

> printf("Total no.of probes for CMBBSHL = %d\n", CNT):
Total no.of probes for CMBBSHL = 44
```

After choosing an evaluation point $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$ (for details of how $\boldsymbol{\alpha}$ should be chosen, see Section 3.2), the program first interpolates $a(\alpha_1, x_2, \alpha_3)$ from the modular black box $\mathbf{B}$ with Chinese remaindering. We got $a(\alpha_1, x_2, \alpha_3) = x_2^2 + 9428x_2 + 18554571$ in the above example. Then it is factored over $\mathbb{Z}$, which gives $(x_2 + 6629)(x_2 + 2799)$. By Hilbert's irreducibility theorem [21], with high probability (see Section 3.3), the irreducible factors of a remain irreducible when evaluated at $\boldsymbol{\alpha}$. Thus, the number of factors of $a(\alpha_1, x_2, \alpha_3)$ is two w.h.p. After two Hensel lifting steps, we obtained the factors `f_pp` $= (x_1+x_2+x_3+1)(x_2+x_3+1)$. These are the factors of the primitive part of a, which equals the final answer `ff`, since a has no content, and no integer content.

Running `CMBBSHLcont` with x_1 to be the main variable gives following:

```
X := [x1, x2, x3]: # Variables with a chosen ordering
alpha := Array(1..3, [2908,3830,2798]): # Evaluation point chosen at random
degA := [1, 2, 2]: # Pre-computed individual degrees
p := prevprime(2^62-1): # A chosen large prime number
Var_Perm := 0: # Yes, X is permuted
Cont_Flag := 1: # Yes, compute and factor the content recursively
sqfinterp := 0: # An option to use SquareFreeImage, turned off
MapleCode := [Maple,C,C,C]: # Use Maple or C code for each subroutine
LI := 0: # Not for large coefficients

> CNT := 0:
> ff := CMBBSHLcont( BBInput, Xnew, alpha, degA, p, Var_Perm, Cont_Flag,
sqfinterp, MapleCode, LI );
```

```
CMBBSHLcont: N = 3
1 prime(s) used to recover a(x1)
N = 3, factors of a(x1) = [[6629*x1+43943641, 1]] in Z[x1]
CMBBSHL step 3: s = 2
fN = [x1+x2+x3+1]
Total time for CMBBSHLcont at N=3 (Computing F_pp) = 0.013000s
CMBBSHLcont: N = 2
1 prime(s) used to recover a(x2)
N = 2, factors of a(x2) = [[x2+2799, 1]]
fN = [x2+x3+1]
Total time for CMBBSHLcont at N=2 (Computing F_pp) = 0.002000s
CMBBSHLcont: N = 1
Total time for CMBBSHLcont at N=1 (deg(a,x3)=0, MakeCont only) = 0.000000s
CMBBSHLcont: N = 0
1 prime(s) used to recover content
Total time for CMBBSHLcont at N=0 = 0.000000s
N = 0, integer content = 1
N = 1, content (in the variable x3) = 1
N = 2, f_pp = x2+x3+1, content (in the variable x2) = 1
N = 3, f_pp = x1+x2+x3+1, content (in the variable x1) = x2+x3+1

               ff := (x1 + x2 + x3 + 1) (x2 + x3 + 1)

> printf("Total no.of probes for CMBBSHL = %d\n", CNT):
Total no.of probes for CMBBSHL = 40
```

In this case, we only got the factor `f_pp` $= x_1 + x_2 + x_3 + 1$ for the primitive part of a. The factors of the content are then computed recursively by building new black boxes from the product of the primitive factors. For a detailed description of content computation, see Section 6.2.

2.2 Computing the irreducible factors of a determinant

Now we consider how to factor the determinant of a 63×63 matrix A of polynomials in $\mathbb{Z}[as, bs, cs, \cdots, is, js, vo]$ (example `heron4d` in Section 6.3). Matrix A comes from a Dixon matrix for solving polynomial system of equations [25]. It is sparse and looks like the

following:

$$\begin{pmatrix}
as^2+gs-hs & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & as^2-ds+fs & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & as^2-bs+cs & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & as^2-ds+fs & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & as^2-bs+cs & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & 0 & -as^2-gs+hs & \cdots \\
 & & & & 0 & 0 & \cdots \\
 & & & & -as^2-gs+hs & 0 & \cdots \\
 & & & & -2as & 0 & \cdots \\
 & & & & 0 & as^2-ds+fs & \cdots \\
 & & & & \vdots & \vdots &
\end{pmatrix}.$$

Let $\tilde{a} = \det(A) \in \mathbb{Z}[as, bs, cs, \cdots, is, js, vo]$. In this case, $n = 11$ and $\det(A)$ has 4 square-free factors. Let $\#f_i$ be the number of terms of each square-free factor, and e_i be the corresponding power of each square-free factor. We have $\#f_i = 1, 6, 22, 131$ and $e_i = 37, 7, 2, 4$. Explicitly,

$$\begin{aligned}
f_1^{e_1} &= as^{37}, \\
f_2^{e_2} &= (as^4 - 2as^2bs - 2as^2cs + bs^2 - 2bs\ cs + cs^2)^7, \\
f_3^{e_3} &= (\underbrace{as^4es + as^2bs\ cs - \cdots - cs\ ds\ fs + cs\ es\ fs}_{22 \text{ terms}})^2, \\
f_4^{e_4} &= (\underbrace{-as^4es^2 + 2as^4es\ is + 2as^4\ es\ js \cdots - fs^2is^2 + 9216\ vo^2}_{131 \text{ terms}})^4.
\end{aligned}$$

If $\tilde{a}$ is expanded out, it has $37,666,243$ terms.

The black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ computes $\det(A(\boldsymbol{\alpha}))$ mod p for a given $\boldsymbol{\alpha} \in \mathbb{Z}^n$ and a prime p. It can be constructed from the matrix A using a Maple procedure `MakeBBdet_C`. The procedure returns an anonymous procedure which is the black box $\mathbf{B}$. The black box $\mathbf{B}$ calls two external C programs from Maple, namely `EVALMOD1` and `Det64`. `EVALMOD1` evaluates the polynomial entries of A at $\boldsymbol{\alpha}$ mod p. `Det64` then computes the modular determinant of the evaluated matrix by Gaussian Elimination.

```
MakeBBdet_C := proc( A::Matrix, VarPerm::list )
    local n,X,N,Xnew,i,AL,AA;
    n := LinearAlgebra:-RowDimension(A);
    X := indets(A); N := nops(X); Xnew := Array(1..N);
    for i to N do Xnew[i] := X[VarPerm[i]]; od;
    Xnew := convert(Xnew,list);
    AL := convert(A,list);
    AA := Array(1..n,1..n,datatype=integer[8],order=C_order);
```

```
    proc( alpha::Array, p::prime ) global CNT,tBBeval,tBBdet;
        local st:=time(),et,d; CNT++;
        EVALMOD1( AL, Xnew, alpha, AA, p );
        et := time()-st; tBBeval += et;
        st := time();
        d := Det64s(AA, n, p);
        et := time()-st; tBBdet += et;
        return d;
    end;
end:
VP := [6, 2, 3, 4, 1, 5, 7, 8, 9, 10, 11] # Choose a variable permutation
BBInput := MakeBBdet_C( A, VP );
```

With trial and error, we have chosen the following variable ordering (corresponding to `VP`) to minimize the computational time:

```
X := [fs, bs, cs, ds, as, es, gs, hs, is, js, vo]:
```

With this ordering, the factors of the primitive part are f_3 and f_4. The factors of the content are f_1 and f_2. Other inputs to procedure `CMBBSHLcont` are:

```
alpha := Array(1..11, [2908, 3830, 2798, 3961, 1324, 2015, 5, 3002, 2347,
         2147, 1573], datatype=integer[8]): # Random evaluation point
degA := [12, 26, 26, 12, 89, 12, 8, 8, 8, 8, 8]: # Pre-computed degrees
p := prevprime(2^62): # A chosen large prime
Var_Perm := 1: # Yes, X is permuted
Cont_Flag := 1: # Yes, compute and factor the content recursively
sqfinterp := 0: # An option to use SquareFreeImage, turned off
MapleCode := [C, C, C, C]: # Use C code for each subroutine
LI := 0: # Not for large coefficients
```

Running `CMBBSHLcont` produces the following output:

```
> CNT := 0: tBBeval := 0: tBBdet := 0:
> st := time():
> ff := CMBBSHLcont( BBInput, X, alpha, degA, p, Var_Perm, Cont_Flag,
        sqfinterp, MapleCode, LI );
35 prime(s) used to recover a(fs)
23 prime(s) used to recover a(bs)
1 prime(s) used to recover a(as)
1 prime(s) used to recover content
N = 0, integer content = 268435456
```

```
N = 1, content (in the variable vo) = 268435456
N = 2, content (in the variable js) = 268435456
N = 3, content (in the variable is) = 268435456
N = 4, content (in the variable hs) = 268435456
N = 5, content (in the variable gs) = 268435456
N = 6, content (in the variable es) = 268435456
N = 7, f_pp = as^37, content (in the variable as) = 268435456
N = 8, content (in the variable ds) = 268435456*as^37
N = 9, content (in the variable cs) = 268435456*as^37
N = 10, f_pp = (as^4-2*as^2*bs-2*as^2*cs+bs^2-2*bs*cs+cs^2)^7, content (in
the variable bs) = 268435456*as^37
N = 11, f_pp = (as^4*es+as^2*bs*cs-as^2*bs*es-as^2*bs*fs-as^2*cs*ds-as^2*cs
*es-as^2*ds*es+as^2*ds*fs+as^2*es^2-as^2*es*fs+bs^2*fs-bs*cs*ds-bs*cs*fs+
bs*ds*es-bs*ds*fs-bs*es*fs+bs*fs^2+cs^2*ds+cs*ds^2-cs*ds*es-cs*ds*fs+cs*es*
fs)^2*(-as^4*es^2+2*as^4*es*is+2*as^4*es*js-as^4*is^2+2*as^4*is*js- as^4*js
^2+4*as^2*bs*cs*js+2*as^2*bs*es*fs-2*as^2*bs*es*gs-2*as^2*bs*es*js-2*as^2
*bs*fs*is-2*as^2*bs*fs*js+2*as^2*bs*gs*is-2*as^2*bs*gs*js-2*as^2*bs*is*js+2
*as^2*bs*js^2+2*as^2*cs*ds*es-2*as^2*cs*ds*is-2*as^2*cs*ds*js-2*as^2*cs*es*
hs-2*as^2*cs*es*js+2*as^2*cs*hs*is-2*as^2*cs*hs*js-2*as^2*cs*is*js+2*as^2*
cs*js^2-2*as^2*ds*es*gs-2*as^2*ds*es*is+4*as^2*ds*fs*is-2*as^2*ds*gs*is+2*
as^2*ds*gs*js+2*as^2*ds*is^2-2*as^2*ds*is*js+2*as^2*es^2*gs+2*as^2*es^2*hs
-2*as^2*es*fs*hs-2*as^2*es*fs*is+4*as^2*es*gs*hs-2*as^2*es*gs*is-2*as^2*es*
gs*js-2*as^2*es*hs*is-2*as^2*es*hs*js+4*as^2*es*is*js-2*as^2*fs*hs*is+2*as
^2*fs*hs*js+2*as^2*fs*is^2-2*as^2*fs*is*js-bs^2*fs^2+2*bs^2*fs*gs+2*bs^2*fs
*js-bs^2*gs^2+2*bs^2*gs*js-bs^2*js^2+2*bs*cs*ds*fs-2*bs*cs*ds*gs-2*bs*cs*ds
*js-2*bs*cs*fs*hs-2*bs*cs*fs*js+2*bs*cs*gs*hs-2*bs*cs*gs*js-2*bs*cs*hs*js+2
*bs*cs*js^2+4*bs*ds*es*gs-2*bs*ds*fs*gs-2*bs*ds*fs*is+2*bs*ds*gs^2-2*bs*ds*
gs*is-2*bs*ds*gs*js+2*bs*ds*is*js-2*bs*es*fs*gs-2*bs*es*fs*hs+2*bs*es*gs^2-
2*bs*es*gs*hs-2*bs*es*gs*js+2*bs*es*hs*js+2*bs*fs^2*hs+2*bs*fs^2*is-2*bs*fs
*gs*hs-2*bs*fs*gs*is+4*bs*fs*gs*js+4*bs*fs*hs*is-2*bs*fs*hs*js-2*bs*fs*is*
js-cs^2*ds^2+2*cs^2*ds*hs+2*cs^2*ds*js-cs^2*hs^2+2*cs^2*hs*js-cs^2*js^2+2*
cs*ds^2*gs+2*cs*ds^2*is-2*cs*ds*es*gs-2*cs*ds*es*hs-2*cs*ds*fs*hs-2*cs*ds*
fs*is-2*cs*ds*gs*hs+4*cs*ds*gs*is-2*cs*ds*gs*js-2*cs*ds*hs*is+4*cs*ds*hs*js
-2*cs*ds*is*js+4*cs*es*fs*hs-2*cs*es*gs*hs+2*cs*es*gs*js+2*cs*es*hs^2-2*cs*
es*hs*js+2*cs*fs*hs^2-2*cs*fs*hs*is-2*cs*fs*hs*js+2*cs*fs*is*js-ds^2*gs^2+2
*ds^2*gs*is-ds^2*is^2+2*ds*es*gs^2-2*ds*es*gs*hs-2*ds*es*gs*is+2*ds*es*hs
*is+2*ds*fs*gs*hs-2*ds*fs*gs*is-2*ds*fs*hs*is+2*ds*fs*is^2-es^2*gs^2+2*es^2
*gs*hs-es^2*hs^2-2*es*fs*gs*hs+2*es*fs*gs*is+2*es*fs*hs^2-2*es*fs*hs*is-fs
^2*hs^2+2*fs^2*hs*is-fs^2*is^2+9216*vo^2)^4, content (in the variable fs) =
```

```
268435456*as^37*(as^4-2*as^2*bs-2*as^2*cs+bs^2-2*bs*cs+cs^2)^7
> et := time() - st:
> printf("Total time for CMBBSHL = %fs\n", et);
Total time for CMBBSHL = 17.559000s
> printf("Total no.of probes for CMBBSHL = %d\n", CNT):
Total no.of probes for CMBBSHL = 68295
```

It took 17.559 seconds to compute the factors of $\det(A)$ with our chosen variable ordering $[fs, bs, cs, ds, as, es, gs, hs, is, js, vo]$. Computing the primitive factors (f_3 and f_4) took 88.9% of the total time (15.616 seconds). 11.1% of the time was to compute the factors of the content recursively.

If we set `Cont_Flag := 0`, then the program only computes the primitive factors f_3 and f_4, and the content is not computed and factored. When solving a parametric system of polynomial equations using resultant methods, one often eliminates variables $x_2, \cdots, x_n$. We will have a resultant $R_1 := R(x_1, y_1, \cdots, y_m)$ for some parameters $y_1, \cdots, y_m$, and we want to solve $R_1 = 0$. In this application we do not want $\text{cont}(R_1, x_1)$ [25].

With different variable orderings, timings can be quite different. For example, if $X = [as, bs, cs, ds, es, fs, gs, hs, is, js, vo]$, the total time was 43.8 seconds (see Table 6.6).

Using the Maple command `LinearAlgebra:-Determinant(A)` only took 0.383 seconds to compute $\det(A)$ as Maple likely recognized that A has a diagonal block structure. Maple returns $\det(A)$ in the form $\det(A) = c_0 g_1 g_2 g_3 g_4$ where

$$
\begin{aligned}
c_0 &= 268435456, \\
g_1 &= f_1^{37} = as^{37}, \\
g_2 &= (f_2^2 f_4)^2 \text{ (in the factored form)}, \\
g_3 &= \text{expand}(f_2 f_4) \ (\#g_3 = 4378), \\
g_4 &= \text{expand}(f_2^2 f_3 f_4) \ (\#g_4 = 8752).
\end{aligned}
$$

To prevent Maple recognizing the block structure, I tried Gentleman and Johnson's algorithm [26] that uses a bottom up minor expansion and Maple ran out of memory. For more benchmarks, see Section 6.3.

All matrices used for our benchmarks are available at:
`http://www.cecm.sfu.ca/~mmonagan/code/BBfactor/`

Chapter 3

Tools for sparse Hensel lifting

In this chapter we present essential tools for designing and analyzing our sparse Hensel lifting algorithms CMSHL and CMBBSHL.

Sections 3.1 to 3.3 are tools for prior Hensel lifting steps. Section 3.1 presents an algorithm to pre-compute the individual degrees of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ represented by a modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$. It is for CMBBSHL. Section 3.2 and 3.3 define **Hilbertian point** and the **weak SHL assumption** which are two necessary conditions for the initial evaluation point $\boldsymbol{\alpha}$ to succeed both CMSHL and CMBBSHL.

Sections 3.4, 3.5 and 3.6 are **square-free factorization**, **bivariate Hensel lifting** and **solving Vandermonde systems** respectively. These are sub-algorithms used at each Hensel lifting step in both CMSHL and CMBBSHL.

Section 3.7 presents **rational number reconstruction** that is used after the last Hensel lifting step to recover the rational coefficients of the factors from their modular images. It is for the non-monic case of algorithm CMBBSHL.

3.1 Computing the individual degrees of a polynomial

The individual degrees of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ (i.e. $\deg(a, x_j)$ for $1 \leq j \leq n$) are needed as an input for CMBBSHL. Given a modular black box representation $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ of a, one can compute the individual degrees of a probabilistically using Algorithm 4.

Algorithm 4 assumes a known (correct) total degree bound D, or individual degree bounds D_j if known. The total degree of a can be computed w.h.p. by Algorithm 3. For the case of computing the determinant of a matrix A with multivariate polynomial entries, the individual degree bounds D_j can be found in the following way.

Let A be an $N \times N$ matrix with $A_{ik} \in \mathbb{Z}[x_1, \cdots, x_n]$ for $1 \leq i \leq N$ and $1 \leq k \leq N$. Then,

$$D_j = \sum_{i=1}^{N} \max_{k=1}^{N}(\deg(A_{ik}), x_j) \geq \deg(\det(A), x_j) = d_j. \tag{3.1}$$

Algorithm 4 Pre-computing $\deg(a, x_j)$.

Input: A modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ representing $a \in \mathbb{Z}[x_1, \cdots, x_n]$, $j \in \mathbb{Z}$ s.t. $1 \leq j \leq n$, individual degree bounds D_j for $1 \leq j \leq n$, a large prime p.
Output: $\deg(a, x_j)$ with high probability (w.h.p.)

1: Pick $\boldsymbol{\beta} = (\beta_1, \cdots, \beta_{j-1}, \beta_{j+1}, \cdots, \beta_n) \in {\mathbb{Z}_p}^{n-1}$ at random.
2: **for** k from 0 to D_j **do**
3: $\quad b_k \leftarrow \mathbf{B}([\beta_1, \cdots, \beta_{j-1}, k, \beta_{j+1}, \cdots, \beta_n], p)$.
4: **end for**
5: Interpolate $h_j(z) \in \mathbb{Z}_p[z]$ from $b_k = h_j(k)$ for $0 \leq k \leq D_j$.
6: **return** $\deg(h_j)$

The failure probability for Algorithm 4 is stated in Proposition 3.1.1.

Proposition 3.1.1. *Let H_j be the smallest absolute value of the coefficients of $\mathrm{LC}(a, x_j)$. Let $d = \deg(a)$ and $d_j = \deg(a, x_j)$. Suppose p is a prime chosen at random from $\mathbb{P}_{63}$. Assume the individual degree bounds D_j's are correct, i.e. $D_j \geq d_j$ for all $1 \leq j \leq n$. Let $\deg(h_j)$ be the output from Algorithm 4. Then,*

$$\Pr[\deg(h_j) < d_j] \leq \frac{d - d_j}{p} + \left\lfloor \frac{\log_2(H_j)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}. \tag{3.2}$$

Proof. Let $h_j(z) := a(\beta_1, \cdots, \beta_{j-1}, z, \beta_{j+1}, \cdots, \beta_n) \bmod p$.
Let $a_{jz} = a(x_1, \cdots, x_{j-1}, z, x_{j+1}, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_{j-1}, x_{j+1}, \cdots, x_n][z]$.
For example, $a = (91x_1^2 + x_3 + 2)x_2^2 + (68x_1x_3 - 10x_3^2 + 3)x_2 - 40x_1^2x_3$.
Let $j = 2$, then $a_{jz} = (91x_1^2 + x_3 + 2)z^2 + (68x_1x_3 - 10x_3^2 + 3)z - 40x_1^2x_3$ and
$\mathrm{LC}(a_{jz}) = 91x_1^2 + x_3 + 2 = \mathrm{LC}(a, x_j)$.
Let $p = 2^{63} - 25$, then $h_j(z) = (91\beta_1^2 + \beta_3 + 2)z^2 + (68\beta_1\beta_3 - 10\beta_3^2 + 3)z - 40\beta_1^2\beta_3$.
We see that

$$\begin{aligned} \deg(h_j) < d_j &\iff 91\beta_1^2 + \beta_3 + 2 = 0 \bmod p \\ &\iff \mathrm{LC}(a, x_j)(\boldsymbol{\beta}) = 0 \bmod p. \end{aligned}$$

This happens if $p | \mathrm{LC}(a, x_j)$ or $p \nmid \mathrm{LC}(a, x_j)$ and $\mathrm{LC}(a, x_j)(\boldsymbol{\beta}) \bmod p = 0$.
H_j has at most $\lfloor \log_2(H_j)/62 \rfloor$ prime divisors from $\mathbb{P}_{63}$. Thus,

$$\Pr[p | \mathrm{LC}(a, x_j)] \leq \Pr[p | H_j] \leq \left\lfloor \frac{\log_2(H_j)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}.$$

Furthermore,

$$\begin{aligned}\Pr[\deg(h_j) < d_j] &= \Pr[\mathrm{LC}(a, x_j)(\boldsymbol{\beta}) = 0 \bmod p \mid p \nmid \mathrm{LC}(a, x_j)] + \Pr[p | \mathrm{LC}(a, x_j)] \\ &\leq \underbrace{\frac{\deg(\mathrm{LC}(a, x_j))}{p}}_{\text{by Lemma 1.6.4}} + \left\lfloor \frac{\log_2(H_j)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|} \\ &\leq \frac{d - d_j}{p} + \left\lfloor \frac{\log_2(H_j)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|}.\end{aligned}$$

□

3.2 Hilbertian point

Prior to sparse Hensel lifting, an evaluation point $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$ is chosen randomly from $[1, \tilde{N} - 1]^{n-1}$ where $\tilde{N} \in \mathbb{Z}^+$ and $\tilde{N} < p$. The initial evaluation point $\boldsymbol{\alpha}$ must satisfy the following two conditions for algorithm CMSHL and CMBBSHL to succeed:

1. $\boldsymbol{\alpha}$ is Hilbertian;
2. $\boldsymbol{\alpha}$ satisfies the weak SHL assumption (see Section 3.3).

Definition 3.2.1. Let $P \in \mathbb{Z}[x_1, \cdots, x_n]$ be an irreducible polynomial over $\mathbb{Z}$. We call a point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ **Hilbertian** if $P(x_1, \alpha_2, \cdots, \alpha_n)$ remains irreducible over $\mathbb{Z}$ and $\deg(P(x_1, \boldsymbol{\alpha})) = \deg(P, x_1)$ [36].

The sharpest result on a bound for the number of non-Hilbertian points of $P(x_1, \cdots, x_n)$ was obtained by Cohen [11], and stated in [51]:

Proposition 3.2.2. *Let $R(d, n, \tilde{N})$ be the number of non-Hilbertian points $(\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ with $0 \leq \alpha_i < \tilde{N}$ ($\tilde{N} \in \mathbb{Z}^+$) for an irreducible polynomial $P(x_1, \cdots, x_n)$ of total degree d. Then,*

$$R(d, n, \tilde{N}) < \bar{c}(d) \tilde{N}^{n-3/2} \log(\tilde{N}), \tag{3.3}$$

where $\bar{c}$ depends only on the degree of the irreducible polynomial.

It is conjectured in [51] that $\bar{c}(d) < c_1 d^{c_2}$ for some absolute constants c_1, c_2. We can see that $\lim_{\tilde{N} \to \infty} R(d, n, \tilde{N}) / \tilde{N}^{n-1} = 0$. Thus, a sufficiently large $\tilde{N}$ ensures $\boldsymbol{\alpha}$ chosen randomly from $[1, \tilde{N} - 1]^{n-1}$ is Hilbertian with high probability.

Example 7. Let $P = x^3 + y^3 + 1 \in \mathbb{Z}[x, y]$. The only non-Hilbertian points are $y = 0$ and $y = -1$.

We remark that in practice non-Hilbertian points are rare.

3.3 The weak SHL assumption

The evaluation point $\boldsymbol{\alpha}$ must also satisfy **the weak SHL assumption** (the weak sparse Hensel lifting assumption) [8, 46] for each factor at every Hensel lifting step. Algorithm CMSHL (and CMBBSHL) differs from algorithm MTSHL [43] as MTSHL uses the **strong SHL assumption** [43, 46] instead.

Definition 3.3.1. Let $\alpha_j \in \mathbb{Z}_p$ be chosen at random. Let

$$f = \sum_{i=0}^{d_j} \sigma_i(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^i \in \mathbb{Z}_p[x_1, \cdots, x_j]$$

be the Taylor polynomial of $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$ about α_j of degree $d_j = \deg(f, x_j)$. Let $\text{Supp}(\sigma_i)$ be the set of all monomials in σ_i. The assumption that $\text{Supp}(\sigma_i) \subseteq \text{Supp}(\sigma_0)$ for all $1 \leq i \leq d_j$ is called **the weak SHL assumption** [8, 46]. The assumption that $\text{Supp}(\sigma_i) \subseteq \text{Supp}(\sigma_{i-1})$ for all $1 \leq i \leq d_j$ is called **the strong SHL assumption** [43, 46].

The following Lemma (stated in [8], modified from Lemma 1 by Monagan and Tuncer [43]) ensures that algorithms CMSHL and CMBBSHL succeed with high probability (w.h.p.). If p is large, it says that the weak SHL assumption holds for almost all choices of $\alpha_j \in \mathbb{Z}_p$.

Lemma 3.3.2. *Let $f \in \mathbb{Z}_p[x_1, \cdots, x_j]$. Let $f = \sum_{i=0}^{d_j} \sigma_i(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^i$ where $d_j = \deg(f, x_j)$. If α_j is a randomly chosen element in $\mathbb{Z}_p$, then*

$$\Pr[\text{Supp}(\sigma_i) \nsubseteq \text{Supp}(\sigma_0)] \leq |\text{Supp}(\sigma_i)| \frac{d_j}{p - d_j + i} \text{ for } 1 \leq i \leq d_j.$$

In other words, the weak SHL assumption assumes that all monomials in the coefficients of $(x_j - \alpha_j)^i$ are contained in $\text{Supp}(\sigma_0)$ for all $1 \leq i \leq d_j$.

Example 8. Let $p = 2^{31} - 1$. One of the factors of the determinant of symmetric Toeplitz matrix T_4 is $f = x_1^2 - x_1x_2 - x_1x_4 - x_2^2 + 2x_2x_3 + x_2x_4 - x_3^2 \in \mathbb{Z}_p[x_1, x_2, x_3, x_4]$. If $\alpha_4 = 3$, expanding f about $x_4 = 3$ gives

$$f = \sigma_0 + \sigma_1(x_4 - \alpha_4) = (x_1^2 - x_1x_2 - x_2^2 + 2x_2x_3 - x_3^2 - 3x_1 + 3x_2) + (-x_1 + x_2)(x_4 - 3).$$

Here, $\text{Supp}(\sigma_0) = \{x_1^2, x_1x_2, x_2^2, x_2x_3, x_3^2, x_1, x_2\}$ and $\text{Supp}(\sigma_1) = \{x_1, x_2\} \subseteq \text{Supp}(\sigma_0)$. The weak SHL assumption is satisfied. However, if we choose $\alpha_4 = 0$, then

$$f = \sigma_0 + \sigma_1(x_4) = (x_1^2 - x_1x_2 - x_2^2 + 2x_2x_3 - x_3^2) + (-x_1 + x_2)x_4.$$

In this case, $\text{Supp}(\sigma_0) = \{x_1^2, x_1x_2, x_2^2, x_2x_3, x_3^2\}$ and $\text{Supp}(\sigma_1) = \{x_1, x_2\}$. The weak SHL assumption fails since $\text{Supp}(\sigma_1) \nsubseteq \text{Supp}(\sigma_0)$.

3.4 Square-free factorization

Square-free factorization is an important technique that is used in designing both algorithms CMSHL and CMBBSHL. For CMSHL, the polynomial to be factored is in its sparse representation and square-free factorization is done prior to Hensel lifting to remove any repeated factors (see Chapter 4 for details). On the other hand, for CMBBSHL, the input polynomial is represented by a black box. Square-free factorization is performed within each Hensel lifting step to compute the *square-free part* of the bivariate images of the input polynomial (see Chapter 6 for details).

The definition for square-free factorization is stated below (see Definition 8.1 in [19]):

Definition 3.4.1. Let $a(x) \in \mathrm{R}[x]$ be a primitive polynomial over a unique factorization domain R (R can be a multivariate polynomial domain, e.g. $\mathbb{Z}[x_2, \cdots, x_n]$). Then $a(x)$ is **square-free** if it has no repeated factors, i.e. there exists no $b(x)$ with $\deg(b(x)) \geq 1$ such that $b(x)^2 | a(x)$. The **square-free factorization** of $a(x)$ is

$$a(x) = \prod_{i=1}^{k} a_i(x)^i \tag{3.4}$$

where each $a_i(x)$ is a square-free polynomial and $\deg(\gcd(a_i(x), a_j(x)), x) = 0$ for $i \neq j$. The square-free factorization is unique up to units. The **square-free part** of $a(x)$, $\mathrm{sqf}(a(x))$ is then defined as

$$\mathrm{sqf}(a(x)) = \prod_{i=1}^{k} a_i(x), \tag{3.5}$$

the product of the square-free factors without their multiplicities.

For our application, we must consider the case where $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$ is non-primitive. Let $h = \mathrm{cont}(a) \in \mathbb{Z}[x_2, \cdots, x_n]$. Consider the factorization

$$a = h \underbrace{\prod_{i=1}^{r} f_i^{e_i}}_{\mathrm{pp}(a)} \tag{3.6}$$

where $\deg(f_i, x_1) > 0$, $\gcd(f_i, f_j) = 1$ for $i \neq j$ and f_i is irreducible in $\mathbb{Z}[x_1, \cdots, x_n]$.

Lemma 3.4.2. *Let $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$ be non-primitive. Assume a has the factorization as in (3.6). Let $g = \gcd(a, \partial a/\partial x_1)$. Then,*
(i) $g = \pm h \prod_{i=1}^{r} f_i^{e_i - 1}$,
(ii) $a/g = \pm \prod_{i=1}^{r} f_i = \pm \mathrm{sqf}(\mathrm{pp}(a))$ *and*
(iii) $\mathrm{cont}(a/g) = \pm 1$.

Proof. (i) From (3.6),

$$\begin{aligned} g &= \gcd\left(a, \frac{\partial a}{\partial x_1}\right) \\ &= \gcd\left(h\prod_{i=1}^{r} f_i^{e_i}, h\left(\sum_{i=1}^{r} e_i f_i^{e_i-1}\frac{\partial f_i}{\partial x_1}\prod_{j\neq i} f_j^{e_j}\right)\right) \\ &= \pm h\prod_{i=1}^{r} f_i^{e_i-1}\cdot\gcd\left(\prod_{i=1}^{r} f_i, \sum_{i=1}^{r} e_i\frac{\partial f_i}{\partial x_1}\prod_{j\neq i} f_j\right). \end{aligned}$$

Since $\gcd(f_i, \partial f_i/\partial x_1) = 1$ and $\gcd(f_i, f_j) = 1$ for $i \neq j$, we have

$$\gcd\left(\prod_{i=1}^{r} f_i, \sum_{i=1}^{r} e_i\frac{\partial f_i}{\partial x_1}\prod_{i\neq j} f_j\right) = \prod_{i=1}^{r}\gcd\left(f_i, \sum_{i=1}^{r} e_i\frac{\partial f_i}{\partial x_1}\prod_{i\neq j} f_j\right) = 1.$$

Thus

$$g = \pm h\prod_{i=1}^{r} f_i^{e_i-1}.$$

(ii) From (i),

$$\frac{a}{g} = \frac{h\prod_{i=1}^{r} f_i^{e_i}}{\pm h\prod_{i=1}^{r} f_i^{e_i-1}} = \pm\prod_{i=1}^{r} f_i.$$

Since each f_i is irreducible, it is square-free. By (3.5), $\prod_{i=1}^{r} f_i = \mathrm{sqf}(\mathrm{pp}(a))$. Thus

$$\frac{a}{g} = \pm\prod_{i=1}^{r} f_i = \pm\mathrm{sqf}(\mathrm{pp}(a)).$$

(iii) Since $\mathrm{sqf}(\mathrm{pp}(a))$ is primitive, $\mathrm{cont}(\mathrm{sqf}(\mathrm{pp}(a)) = 1$. Therefore,

$$\mathrm{cont}(a/g) = \mathrm{cont}(\pm\mathrm{sqf}(\mathrm{pp}(a)) = \pm\mathrm{cont}(\mathrm{sqf}(\mathrm{pp}(a)) = \pm 1.$$

□

Definition 3.4.3. Let $a \in \mathbb{Z}[x_2, \cdots, x_n][x_1]$ (not necessarily primitive). We define the square-free part of a, denoted as $\mathrm{sqf}(a)$ to be the square-free part of the primitive part of a, i.e.

$$\mathrm{sqf}(a) = \mathrm{sqf}(\mathrm{pp}(a)). \tag{3.7}$$

In our factorization algorithm CMBBSHL, we will do many Hensel lifts with $\mathrm{sqf}(a)$ for $a \in \mathbb{Z}_p[x_1, x_j]$. We use Lemma 3.4.2 part (ii) to compute $\mathrm{sqf}(a)$ and it is unique up to units.

Example 9. Consider $a = -3(2y+z)(x+y+z)^2(x^2+21y+5)^3 \in \mathbb{Z}[y,z][x]$. Then,

$$g = \gcd\left(a, \frac{\partial a}{\partial x}\right) = 3(2y+z)(x+y+z)(x^2+21y+5)^2.$$

Thus,

$$\frac{a}{g} = -(x+y+z)(x^2+21y+5) = -\mathrm{sqf}(\mathrm{pp}(a)) = -\mathrm{sqf}(a).$$

Both the polynomial content $2y+z$ and the integer content 3 of a are removed in $\mathrm{sqf}(a)$.

A consequence of using $\mathrm{sqf}(a) = \pm a/\gcd(a, \partial a/\partial x_1)$ to lift the irreducible factors of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ is that we lose $\mathrm{cont}(a)$. In applications where we do not need $\mathrm{cont}(a)$, this will be a significant computational advantage of our factorization algorithm CMBBSHL when $\mathrm{cont}(a)$ is larger than $\mathrm{pp}(a)$. Table 6.3 shows an application that illustrates that the time to factor $\mathrm{pp}(a)$ is 10 times smaller than the time to factor $\mathrm{cont}(a)$.

3.5 Bivariate Hensel lifting

Many bivariate Hensel lifts (BHL) are performed in each Hensel lifting step in algorithms CMSHL and CMBBSHL. Since both CMSHL and CMBBSHL are modular algorithms, bivariate Hensel lifting is for lifting polynomials (lifting univariate images to bivariate images) with coefficients in $\mathbb{Z}_p$. Below we present a quintic algorithm, followed by Bernardin's quartic algorithm [4], and then a high-level explanation of Monagan and Paluck's cubic algorithm [48]. For simplicity, we first consider the monic case, i.e. $\mathrm{LC}(a,x) = 1$ (x is the chosen main variable). The non-monic case can be modified accordingly (see Section 6.1.1). All BHL algorithms presented in this thesis are *linear* Hensel lifting algorithms.

Given $a \in \mathbb{Z}_p[x,y]$ where a is square-free and monic in x, i.e. $\mathrm{LC}(a,x) = 1$. Define $d_x = \deg(a,x)$ and $d_y = \deg(a,y)$. Given some $\alpha \in \mathbb{Z}_p$ and $f_{1,0}, \cdots, f_{r,0} \in \mathbb{Z}_p[x]$ s.t.

(i) $\gcd(f_{k,0}, f_{l,0}) = 1$ for $k \neq l$,

(ii) $a(y=\alpha) = \prod_{\rho=1}^{r} f_{\rho,0}$,

(iii) $f_{\rho,0}$ is monic for $1 \leq \rho \leq r$.

The goal of bivariate Hensel lifting is to construct $f_\rho \in \mathbb{Z}_p[x,y]$ (assuming f_ρ exists) for $1 \leq \rho \leq r$ s.t.

(i) $a = \prod_{\rho=1}^{r} f_\rho \bmod (y-\alpha)^k$ for any $k \geq 1$,

(ii) $f_\rho(y=\alpha) = f_{\rho,0}$ for $1 \leq \rho \leq r$,

(iii) f_ρ is monic in x for $1 \leq \rho \leq r$.

Algorithm 5 Bivariate Hensel lifting: monic case – $\mathcal{O}(d_x^2 d_y^3)$

Input: A prime p, $\alpha \in \mathbb{Z}_p$, $a \in \mathbb{Z}_p[x,y]$ where a is square-free and monic in x, i.e. $\mathrm{LC}(a,x) = 1$, $f_{\rho,0} \in \mathbb{Z}_p[x]$ for $1 \le \rho \le r$ s.t. (i) $\gcd(f_{k,0}, f_{l,0}) = 1$ for $k \neq l$, (ii) $a(y=\alpha) = \prod_{\rho=1}^{r} f_{\rho,0}$, (iii) $f_{\rho,0}$ is monic for $1 \le \rho \le r$.
Output: $f_\rho \in \mathbb{Z}_p[x,y]$ for $1 \le \rho \le r$ s.t. (i) $a = \prod_{\rho=1}^{r} f_\rho$, (ii) $f_\rho(y=\alpha) = f_{\rho,0}$ for all $1 \le \rho \le r$, (iii) f_ρ is monic in x for $1 \le \rho \le r$. Or FAIL (f_ρ does not exists, for some $1 \le \rho \le r$).

1: $dx \leftarrow \deg(a,x); dy \leftarrow \deg(a,y); df_{\rho,0} \leftarrow \deg(f_{\rho,0},x)$.
2: $M \leftarrow \prod_{\rho=1}^{r} f_{\rho,0} \in \mathbb{Z}_p[x]$.
3: **for** ρ from 1 to r **do** $f_\rho \leftarrow f_{\rho,0}$; $M_\rho \leftarrow M/f_{\rho,0}$ **end for**
4: **for** k from 1 to dy **do**
5: $\quad ac_k \leftarrow \mathrm{coeff}(a,(y-\alpha)^k) \in \mathbb{Z}_p[x]$.
6: $\quad \Delta_k \leftarrow \mathrm{coeff}(\prod_{\rho=1}^{r} f_\rho,(y-\alpha)^k) \in \mathbb{Z}_p[x]$.
7: $\quad c_k \leftarrow ac_k - \Delta_k \in \mathbb{Z}_p[x]$.
8: $\quad$ **if** $\sum_{\rho=1}^{r} \deg(f_\rho,y) = dy$ and $c_k \neq 0$ **return FAIL end if**
9: $\quad$ **if** $c_k \neq 0$ **then**
10: $\quad\quad$ Solve $\sum_{\rho=1}^{r} \sigma_{\rho,k} M_\rho = c_k$ for $\sigma_{\rho,k} \in \mathbb{Z}_p[x]$ with $\deg(\sigma_{\rho,k},x) < \deg(\sigma_{\rho,0},x)$ for $1 \le \rho \le r$. // a multi-term Diophantine equation
11: $\quad\quad$ **for** ρ from 1 to r **do** $f_\rho \leftarrow f_\rho + \sigma_{\rho,k}(x)(y-\alpha)^k$ **end for**
12: $\quad$ **end if**
13: **end for**
14: **if** $c_k \neq 0$ **and** $a - \prod_{\rho=1}^{r} f_\rho \neq 0$ **then return FAIL end if**
15: **return** f_ρ for $1 \le \rho \le r$.

We first present a quintic algorithm which does $\mathcal{O}(d_x^2 d_y^3)$ multiplications in $\mathbb{Z}_p$ in Algorithm 5. Let

$$f_\rho^{(k)} = \sum_{j=0}^{k-1} \sigma_{\rho,j}(y-\alpha)^j \tag{3.8}$$

for all $1 \le \rho \le r$ with $\sigma_{\rho,j} \in \mathbb{Z}_p[x]$. If f_ρ exists for all $1 \le \rho \le r$, then the uniqueness of $f_\rho^{(k)}$ is also guaranteed at each $k \ge 1$. This is because the solution $\sigma_{\rho,k}$ of the Diophantine equation in step 10 is unique for each $k \ge 1$ (Theorem 2.6 in [19]).

The bottleneck of Algorithm 5 is at Step 6. In the computation of

$$\Delta_k = \mathrm{coeff}\left(\prod_{\rho=1}^{r} f_\rho^{(k)}, (y-\alpha)^k\right), \tag{3.9}$$

computing the product $\prod_{\rho=1}^{r} f_\rho^{(k)}$ is the bottleneck. Naively, we can compute $\prod_{\rho=1}^{r} f_\rho^{(k)}$ in (3.9) by multiplying the factors one at a time. Since

$$f_1^{(k)} = \sigma_{1,0}(x) + \sigma_{1,1}(x)(y-\alpha) + \sigma_{1,2}(x)(y-\alpha)^2 + \cdots + \sigma_{1,k-1}(x)(y-\alpha)^{k-1},$$
$$f_2^{(k)} = \sigma_{2,0}(x) + \sigma_{2,1}(x)(y-\alpha) + \sigma_{2,2}(x)(y-\alpha)^2 + \cdots + \sigma_{2,k-1}(x)(y-\alpha)^{k-1},$$

multiplying $f_1^{(k)}$ and $f_2^{(k)}$ needs at most k^2 multiplications of $\sigma_{1,i}$ and $\sigma_{2,j}$'s. Now let

$$\begin{aligned} P_2^{(k)} &= f_1^{(k)} f_2^{(k)} \bmod (y-\alpha)^{k+1} \\ &= p_{2,0}(x) + p_{2,1}(x)(y-\alpha) + p_{2,2}(x)(y-\alpha)^2 + \cdots + p_{2,k}(x)(y-\alpha)^k, \end{aligned}$$

and then

$$\begin{aligned} P_3^{(k)} &= P_2^{(k)} f_3^{(k)} \bmod (y-\alpha)^{k+1} \\ &= p_{3,0}(x) + p_{3,1}(x)(y-\alpha) + p_{3,2}(x)(y-\alpha)^2 + \cdots + p_{3,k}(x)(y-\alpha)^k. \end{aligned}$$

Multiplying $P_2^{(k)}$ and $f_3^{(k)}$ now needs at most $k(k+1)$ multiplications of $p_{2,i}$ and $\sigma_{3,j}$'s. Let $d_{f_{\rho,0}} = \deg(f_{\rho,0}, x)$ for $1 \le \rho \le r$, we have $\deg(p_{2,i}, x) \le d_{f_{1,0}} + d_{f_{2,0}}$ for all $i = 0, \cdots, k$. Assuming classical polynomial multiplications, the costs of computing $P_2^{(k)}$ and $P_3^{(k)}$ are at most $k^2 d_{f_{1,0}} d_{f_{2,0}}$ and $k(k+1)(d_{f_{1,0}} + d_{f_{2,0}}) d_{f_{3,0}}$ multiplications in $\mathbb{Z}_p$ respectively.

Define in general

$$P_q^{(k)} = \prod_{\rho=1}^{q} f_\rho^{(k)} \bmod (y-\alpha)^{k+1}, \tag{3.10}$$

for $2 \le q \le r$. For each k in the loop (for $1 \le k \le d_y$), computing $P_r^{(k)}$ costs at most

$$\sum_{i=1}^{r-1} k(k+1) i d_{f_{\max,0}}^2 = \frac{r(r-1)}{2} k(k+1) d_{f_{\max,0}}^2 < \frac{1}{2} k(k+1) d_x^2$$

multiplications in $\mathbb{Z}_p$. In the above, $d_{f_{\max,0}} = \max_{\rho=1}^{r} d_{f_{\rho,0}}$, $d_x = \deg(a, x)$ and $d_y = \deg(a, y)$. We used the inequality $r d_{f_{\max,0}} \le d_x$.

Therefore, the total cost of Algorithm 5 is at most

$$\sum_{k=1}^{d_y} \frac{1}{2} k(k+1) d_x^2 = \frac{1}{6} d_x^2 d_y (d_y^2 + 3d_y + 2) \in \mathcal{O}\left(d_x^2 d_y^3\right) \tag{3.11}$$

multiplications in $\mathbb{Z}_p$.

3.5.1 Bernardin's quartic algorithm

Bernardin's bivariate Hensel lifting algorithm [4] costs $\mathcal{O}(d_x^2 d_y^2)$. Instead of naively multiplying the factors to compute $P_q^{(k)}$, it uses the terms computed in the previous step to save computations.

The product of the first two factors gives

$$\begin{aligned} P_2^{(k)} &= \sigma_{1,0}\sigma_{2,0} + (\sigma_{1,0}\sigma_{2,1} + \sigma_{1,1}\sigma_{2,0})(y-\alpha) + \cdots \\ &\quad + (\sigma_{1,0}\sigma_{2,k-1} + \cdots + \sigma_{1,k-2}\sigma_{2,1} + \sigma_{1,k-1}\sigma_{2,0})(y-\alpha)^{k-1} \\ &\quad + (\sigma_{1,1}\sigma_{2,k-1} + \cdots + \sigma_{1,k-2}\sigma_{2,2} + \sigma_{1,k-1}\sigma_{2,1})(y-\alpha)^{k}. \end{aligned}$$

For successive q $(3 \le q \le r)$, $P_q^{(k)}$ can be computed as

$$\begin{aligned} P_q^{(k)} &= p_{q-1,0}\sigma_{q,0} + (p_{q-1,0}\sigma_{q,1} + p_{q-1,1}\sigma_{q,0})(y-\alpha) + \cdots \\ &\quad + (p_{q-1,0}\sigma_{q,k-1} + \cdots + p_{q-1,k-2}\sigma_{q,1} + p_{q-1,k-1}\sigma_{q,0})(y-\alpha)^{k-1} \\ &\quad + (p_{q-1,1}\sigma_{q,k-1} + \cdots + p_{q-1,k-1}\sigma_{q,1} + p_{q-1,k}\sigma_{q,0})(y-\alpha)^{k}, \end{aligned}$$

where $p_{q-1,j} = \text{coeff}(P_{q-1}^{(k)}, (y-\alpha)^j)$. Moving to step $k+1$ we have

$$\begin{aligned} P_2^{(k+1)} &= \sigma_{1,0}\sigma_{2,0} + (\sigma_{1,0}\sigma_{2,1} + \sigma_{1,1}\sigma_{2,0})(y-\alpha) + \cdots \\ &\quad + (\sigma_{1,0}\sigma_{2,k-1} + \cdots + \sigma_{1,k-2}\sigma_{2,1} + \sigma_{1,k-1}\sigma_{2,0})(y-\alpha)^{k-1} \\ &\quad + (\sigma_{1,0}\sigma_{2,k} + \cdots + \sigma_{1,k-1}\sigma_{2,1} + \sigma_{1,k}\sigma_{2,0})(y-\alpha)^{k} \\ &\quad + (\sigma_{1,1}\sigma_{2,k} + \cdots + \sigma_{1,k-1}\sigma_{2,2} + \sigma_{1,k}\sigma_{2,1})(y-\alpha)^{k+1}, \end{aligned}$$

$$\begin{aligned} P_q^{(k+1)} &= p_{q-1,0}\sigma_{q,0} + (p_{q-1,0}\sigma_{q,1} + p_{q-1,1}\sigma_{q,0})(y-\alpha) + \cdots \\ &\quad + (p_{q-1,0}\sigma_{q,k-1} + \cdots + p_{q-1,k-2}\sigma_{q,1} + p_{q-1,k-1}\sigma_{q,0})(y-\alpha)^{k-1} \\ &\quad + (p_{q-1,0}\sigma_{q,k} + \cdots + p_{q-1,k-1}\sigma_{q,1} + p_{q-1,k}\sigma_{q,0})(y-\alpha)^{k} \\ &\quad + (p_{q-1,1}\sigma_{q,k} + \cdots + p_{q-1,k}\sigma_{q,1} + p_{q-1,k+1}\sigma_{q,0})(y-\alpha)^{k+1}, \end{aligned}$$

where $p_{q-1,j} = \text{coeff}(P_{q-1}^{(k+1)}, (y-\alpha)^j)$.

Observe that from step k to step $k+1$, the only terms that need to be computed and have not been computed are

$$\sigma_{1,0}\sigma_{2,k}, \quad \sigma_{1,k}\sigma_{2,0}, \quad \text{and} \quad \sigma_{1,1}\sigma_{2,k} + \cdots + \sigma_{1,k-1}\sigma_{2,2} + \sigma_{1,k}\sigma_{2,1} \tag{3.12}$$

for the first two factors and

$$p_{q-1,0}\sigma_{q,k} \quad \text{and} \quad p_{q-1,1}\sigma_{q,k} + \cdots + p_{q-1,k}\sigma_{q,1} + p_{q-1,k+1}\sigma_{q,0} \tag{3.13}$$

for each subsequent product $P_q^{(k+1)}$.

Assuming polynomial multiplications have quadratic cost, the total cost for step k is

$$\sum_{i=1}^{r-1}(k+2)id_{f_{\max,0}}^2 = \frac{r(r-1)}{2}(k+2)d_{f_{\max,0}}^2 < \frac{1}{2}(k+2)d_x^2$$

multiplications in $\mathbb{Z}_p$. Therefore, the total cost for Bernardin's algorithm is at most

$$\sum_{k=1}^{d_y}\frac{1}{2}(k+2)d_x^2 = \frac{1}{4}d_x^2 d_y(d_y+5) \in \mathcal{O}\left(d_x^2 d_y^2\right) \tag{3.14}$$

multiplications in $\mathbb{Z}_p$.

3.5.2 Monagan and Paluck's cubic algorithm

Monagan and Paluck's BHL algorithm [48] further reduces the cost to $\mathcal{O}(d_x^2 d_y + d_x d_y^2)$ arithmetic operations in $\mathbb{Z}_p$. Instead of multiplying polynomials in $\mathbb{Z}_p[x]$ to compute the missing terms in (3.12) and (3.13) as in Bernardin's algorithm, Monagan and Paluck's algorithm uses evaluation and interpolation to compute (3.12) and (3.13) and hence Δ_k in (3.9). We present their idea with a homomorphism diagram in Figure 3.1 [48].

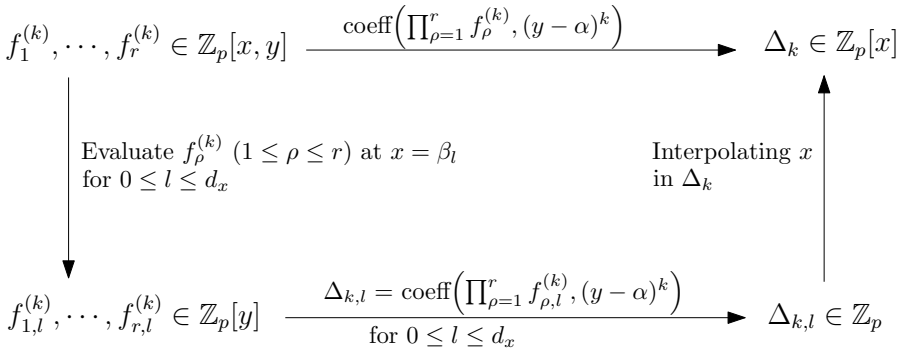

Figure 3.1: Homomorphism diagram for computing Δ_k in Monagan and Paluck's cubic algorithm.

Each $f_\rho^{(k)}$ $(1 \le \rho \le r)$ is evaluated at $x = \beta_l$ for $0 \le l \le d_x$. Then multiplications are performed in $\mathbb{Z}_p$ instead of $\mathbb{Z}_p[x]$. After computing $\Delta_{k,l}$ (the images of Δ_k at $x = \beta_l$), Lagrange interpolation is used to recover $\Delta_k \in \mathbb{Z}_p[x]$.

An earlier version of Monagan and Paluck's algorithm [49] computes the missing terms as exactly in (3.12) and (3.13) by evaluation and interpolation with factors multiplied one at a time. The most recent version [48] further reduces the space complexity so that the factors are no longer multiplied one at a time but as pairs of products instead. For either version of their algorithms, the cost for evaluation and interpolation is $\mathcal{O}(d_x^2 d_y)$ arithmetic operations in $\mathbb{Z}_p$ and the cost of computing $\Delta_{k,l}$ for $1 \le l \le d_x$ is $\mathcal{O}(d_x d_y^2)$ arithmetic operations in $\mathbb{Z}_p$. The total cost is $\mathcal{O}(d_x^2 d_y + d_x d_y^2)$. For detailed complexity analyses, see [48, 49].

3.6 Solving Vandermonde systems

3.6.1 Solving Vandermonde systems in CMSHL

Vandermonde systems of linear equations appear in both algorithms CMSHL and CMBB-SHL. At the j^{th} Hensel lifting step, we evaluate $f_\rho(x_1, \cdots, x_{j-1}, \alpha_j, \cdots, \alpha_n)$ at

$$x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k$$

for $k = 1, 2, 3, \cdots, s$, where s is the number of bivariate images needed at the j^{th} Hensel lifting step and $\beta_2, \cdots, \beta_{j-1}$ are chosen at random from $\mathbb{Z}_p \backslash \{0\}$. The definition of s is defined in step 5 of Algorithm 9. The sizes of the Vandermonde systems s_i at step 15 of Algorithm 9 can be smaller than s, since s is the maximum of the sizes of all those Vandermonde matrices. We present an example below to show how Vandermonde systems appear in CMSHL (also CMBBSHL).

Example 10. Let $p = 101$ (a large prime is usually needed for CMSHL, but in this example $p = 101$ is sufficient) and let $j = 5$. Let

$$f = x_1^4 + (29x_2x_3 + 10x_4^2)x_5^2x_1^3 - 3x_2x_3^2x_5x_1^2 + (8x_2x_3^3 + 4x_2x_3x_4)x_1 \in \mathbb{Z}_{101}[x_1, \cdots, x_5].$$

In this case, we need $s = 2$ bivariate images, $f_1(x_1, x_5)$ and $f_2(x_1, x_5)$, to recover the coefficients 29 and 10 corresponding to the monomials x_2x_3 and x_4^2 in the second term of f.

Suppose $\beta_2 = 15, \beta_3 = 97$, and $\beta_4 = 42$ have been chosen. For $k = 1$, the monomials $M_1 = x_2x_3$ and $M_2 = x_4^2$ are evaluated as $m_1 = \beta_2\beta_3 = 41$ and $m_2 = \beta_4^2 = 47$. For $k = 2$, M_1 and M_2 are evaluated at $[x_2, x_3, x_4] = [\beta_2^2, \beta_3^2, \beta_4^2]$. Note that

$$m_i^k = (M_i(\beta_2, \cdots, \beta_{j-1}))^k = M_i(\beta_2^k, \cdots, \beta_{j-1}^k) \tag{3.15}$$

for all $k \geq 1$ (for $i = 1, \cdots, s$). Evaluating f at $[x_2, x_3, x_4] = [\beta_2^k, \beta_3^k, \beta_4^k]$ for $k = 1, 2$ gives

$$\begin{aligned} f_1 &= x_1^4 + (29 \cdot 41 + 10 \cdot 47)x_1^3x_5^2 + 88x_1^2x_5 + 16x_1 = x_1^4 + 43x_1^3x_5^2 + 88x_1^2x_5 + 16x_1, \\ f_2 &= x_1^4 + (29 \cdot 41^2 + 10 \cdot 47^2)x_1^3x_5^2 + 11x_1^2x_5 + x_1 = x_1^4 + 38x_1^3x_5^2 + 11x_1^2x_5 + x_1. \end{aligned}$$

Note that we avoided the evaluation $[x_2, x_3, x_4] = [\beta_2^0, \beta_3^0, \beta_4^0] = [1, 1, 1]$. This is because we want random evaluation points for the variables $x_2, \cdots, x_{j-1}$ at other steps (e.g. step 8 of Algorithm 13). Thus we get a *shifted* transposed Vandermonde system:

$$\underbrace{\begin{pmatrix} m_1 & m_2 \\ m_1^2 & m_2^2 \end{pmatrix}}_{\tilde{V}} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 43 \\ 38 \end{pmatrix}.$$

Solving it gives $c_1 = 10$ and $c_2 = 29$.

For distinct monomial evaluations m_i, $\det(\tilde{V}) \neq 0$, and we can get a unique solution. Monomial evaluations might not be distinct as β_i's are chosen at random from $\mathbb{Z}_p \backslash \{0\}$.

3.6.2 Solving transposed Vandermonde systems in $\mathcal{O}(s^2)$

We present the quadratic algorithm of Zippel [62] to solve the transposed Vandermonde system. For an $s \times s$ system, Zippel's algorithm does $\mathcal{O}(s^2)$ arithmetic operations in $\mathbb{Z}_p$ and uses $\mathcal{O}(s)$ space of elements of $\mathbb{Z}_p$. For most implementation examples of algorithm CMBBSHL, solving Vandermonde systems is not the bottleneck. For computing the factors of the determinant of Toeplitz matrices, solving the Vandermonde systems is the bottleneck. Thus, I integrated a Maple implementation of the fast Vandermonde solver of Kaltofen and Yagati [32] which costs $\mathcal{O}(M(s) \log s)$.

Given distinct $m_i \in \mathbb{Z}_p$ (p is a prime), $1 \leq i \leq s$ ($s \geq 1$), $\boldsymbol{b} = (b_1, \cdots, b_s)^T \in {\mathbb{Z}_p}^s$, we want to solve for $\boldsymbol{c} = (c_1, \cdots, c_s)^T \in {\mathbb{Z}_p}^s$ s.t.

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ m_1 & m_2 & \cdots & m_s \\ \vdots & \vdots & & \vdots \\ m_1^{s-1} & m_2^{s-1} & \cdots & m_s^{s-1} \end{pmatrix}}_{V} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{pmatrix}. \tag{3.16}$$

Since $\det(V) = \prod_{1 \leq i < j \leq s} (m_i - m_j)$ and $m_i \neq m_j$, we have $\det(V) \neq 0$. We can compute V^{-1} and hence $\boldsymbol{c} = V^{-1}\boldsymbol{b}$. Let

$$V^{-1} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & & & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{ss} \end{pmatrix}.$$

Then,

$$V^{-1}V = I = \begin{pmatrix} P_1(m_1) & P_1(m_2) & \cdots & P_1(m_s) \\ P_2(m_1) & P_2(m_2) & \cdots & P_2(m_s) \\ \vdots & & & \vdots \\ P_s(m_1) & P_s(m_2) & \cdots & P_s(m_s) \end{pmatrix},$$

where $P_j(x) = a_{j1} + a_{j2}x + \cdots + a_{js}x^{s-1}$ for $j = 1, \cdots, s$. We need

$$P_j(m_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases}$$

Let

$$M(x) = \prod_{i=1}^{s}(x - m_i) \text{ and } Q_j(x) = M(x)/(x - m_j). \tag{3.17}$$

$P_j(x)$'s are the Lagrange basis polynomials, i.e.

$$P_j(x) = \frac{(x - m_1)\cdots(x - m_{j-1})(x - m_{j+1})\cdots(x - m_s)}{(m_j - m_1)\cdots(m_j - m_{j-1})(m_j - m_{j+1})\cdots(m_j - m_s)} = \frac{Q_j(x)}{Q_j(m_j)}.$$

$P_j(x)$ can be computed using

$$P_j(x) = Q_j(x)Q_j(m_j)^{-1}. \tag{3.18}$$

The coefficients of $P_j(x)$ are $(a_{j1}, \cdots, a_{js})$, which is row j of V^{-1}. Finally, $\boldsymbol{c} = V^{-1}\boldsymbol{b}$.

Computing $M(x)$ in (3.17) costs $\mathcal{O}(s^2)$ arithmetic operations in $\mathbb{Z}_p$ if using multiplications with quadratic cost. For each j, computing $Q_j(x)$ using ordinary division of $M(x)$ by $x - m_j$ costs $\mathcal{O}(s)$ arithmetic operations in $\mathbb{Z}_p$. Evaluating $Q_j(x)$ at $x = m_j$ using Horner's evaluation costs $\mathcal{O}(s)$ arithmetic operations in $\mathbb{Z}_p$. Thus, computing $P_j(x)$ for $j = 1, \cdots, s$ costs $\mathcal{O}(s^2)$ arithmetic operations in $\mathbb{Z}_p$. The total cost for solving the transposed Vandermonde system (3.16) using the method above is $\mathcal{O}(s^2)$ arithmetic operations in $\mathbb{Z}_p$.

For a *shifted* transposed Vandermonde system

$$\underbrace{\begin{pmatrix} m_1 & m_2 & \cdots & m_s \\ m_1^2 & m_2^2 & \cdots & m_s^2 \\ \vdots & \vdots & & \vdots \\ m_1^{s-1} & m_2^{s-1} & \cdots & m_s^{s-1} \end{pmatrix}}_{\tilde{V}} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{pmatrix}, \tag{3.19}$$

we notice that

$$\tilde{V} = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ m_1 & m_2 & \cdots & m_s \\ \vdots & \vdots & & \vdots \\ m_1^{s-1} & m_2^{s-1} & \cdots & m_s^{s-1} \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} m_1 & & & \\ & m_2 & & \\ & & \ddots & \\ & & & m_s \end{pmatrix}}_{D}$$

with D a diagonal matrix. Since $\tilde{V}\boldsymbol{c} = V\underbrace{(D\boldsymbol{c})}_{\boldsymbol{u}} = \boldsymbol{b}$, we first solve $V\boldsymbol{u} = \boldsymbol{b}$ for $\boldsymbol{u}$ and then $\boldsymbol{c} = D^{-1}\boldsymbol{u}$.

3.7 Rational number reconstruction

For the non-monic case of our black box factorization algorithm CMBBSHL, we use rational number reconstruction after sparse Hensel lifting to recover the rational coefficients from modular images and then the integer coefficients of the factors. The problem of rational number reconstruction is described as follows:

Let $n/d \in \mathbb{Q}$, $\gcd(n,d) = 1$ and $d > 0$. Suppose we have computed $u \equiv n/d \bmod m$, $0 \leq u < m$ and $\gcd(d,m) = 1$. Question: How can we recover n/d from u, m?

The following theorem [57] shows that Wang's algorithm `RATCONVERT` [56] always reconstructs such n/d provided $m > 2ND$ where $N, D \in \mathbb{Z}$, $N \geq |n|$ and $D \geq d$.

Theorem 3.7.1. *(Wang, Guy and Davenport, 1982) Let $n, d \in \mathbb{Z}$ with $d > 0$ and $\gcd(n,d) = 1$. Let $m \in \mathbb{N}$ and $\gcd(m,d) = 1$. Suppose $u \equiv n/d \bmod m$ with $0 \leq u < m$. Let $N, D \in \mathbb{Z}$ s.t. $N \geq |n|$ and $D \geq d$. Then,*
(i) if $m > 2ND$, then for any $0 \leq u < m$ there exists a unique rational number n/d s.t. $n/d \equiv u \bmod m$, and
(ii) if $m > 2ND$ then on input of m, u, there exists a unique index i in the extended Euclidean algorithm s.t. $r_i/t_i = n/d$. Moreover, i is the first index s.t. $r_i \leq N$.

Example 11. Let $n = 22, d = 7$. Let $m = 21311$ and $N = D = 100$ so that $m > 2ND$. We have $22/7 \equiv 6092 \bmod 21311$. Running the extended Euclidean algorithm in Maple gives the following:

```
> EEA := proc(m::integer, u::integer)
  local r, s, t, i, q;
  r[0] := m; r[1] := u;
  s[0] := 1; s[1] := 0;
  t[0] := 0; t[1] := 1;
  printf("%4s %7s %7s %7s %12s\n", "i","r[i]","t[i]","q[i+1]","r[i]/t[i]");
  for i from 1 while ( r[i]<>0 ) do
    q[i+1] := iquo(r[i-1],r[i]);
    s[i+1] := s[i-1] - q[i+1]*s[i];
    r[i+1] := r[i-1] - q[i+1]*r[i];
    t[i+1] := t[i-1] - q[i+1]*t[i];
    printf("%4d %7d %7d %7d %12a\n",i,r[i],t[i],q[i+1],r[i]/t[i]);
  od;
  return r[i-1],s[i-1],t[i-1];
end:

> EEA(21311,6092);
   i    r[i]    t[i]  q[i+1]    r[i]/t[i]
```

```
1     6092        1        3          6092
2     3035       -3        2       -3035/3
3       22        7      137          22/7
4       21     -962        1       -21/962
5        1      969       21         1/969

                  1, -277, 969
```

We see that $i = 3$ is the first index s.t. $r_i = 22 \leq N = 100$. So we recover the rational number $r_3/t_3 = n/d = 22/7$ by Wang's algorithm [56].

Maple's `iratrecon(u,m,N,D)` uses Wang's algorithm by default with specified N, D.

```
> iratrecon(6092,21311,100,100);
                                22/7
```

If N, D are unspecified, `iratrecon(u,m)` uses Wang's algorithm by default with $N = D = k$, where k is the largest integer s.t. $2k^2 < m$.

```
> iratrecon(6092,21311);
                                22/7
```

Maple's `iratrecon` command has an option to use Monagan's Maximal Quotient Rational Reconstruction [38]. It is a probabilistic algorithm that is more efficient than Wang's algorithm especially when $|n| \ll d$ or $d \ll |n|$. Currently our algorithm CMBBSHL uses `iratrecon(u,m)`, i.e. Wang's algorithm without specifying N, D.

Chapter 4

Algorithms for sparse Hensel lifting

In this chapter, we present two algorithms developed by Monagan and Tuncer [43, 45], and then algorithm CMSHL by Monagan and myself [8]. These are *sparse Hensel lifting algorithms* used for factoring sparse polynomials given in the sparse representation. We describe Monagan and Tuncer's algorithms in Section 4.2, and then CMSHL in Section 4.3. The last section gives worst-case complexity analyses for both MTSHL and CMSHL.

We refer to the algorithm developed by Monagan and Tuncer in 2016 as MTSHL [43]. For MTSHL, the authors used the **strong SHL assumption** (Lemma 1 of [43]) to solve the multivariate polynomial Diophantine equations (MDPs) that appear in Wang's MHL algorithm in random polynomial time. A detailed complexity analysis for the average-case was completed in [46]. MTSHL was integrated into Maple 2019 as the default command `factor` [47].

In 2018, Monagan and Tuncer introduced another approach [45]. Instead of solving MDPs, at each Hensel lifting step, the factors are recovered from many bivariate images obtained from bivariate Hensel lifts (BHL). Classical BHL by Bernardin [4] costs $O(d_x^2 d_y^2)$ for an input polynomial $a \in \mathbb{Z}_p[x, y]$ with $d_x = \deg(a, x)$ and $d_y = \deg(a, y)$. The cost of BHL is improved to $O(d_x^2 d_y + d_x d_y^2)$ by Monagan and Paluck [39, 48]. This approach is suitable for sparse Hensel lifting since the individual degrees of the factors are often low, e.g. $\deg(a, x_j) < 1000$ for all $1 \leq j \leq n$.

However, it is observed in both Monagan and Tuncer's algorithms, as well as in Wang's MHL, when the evaluation point α_j is non-zero, an expression swell occurs in each factor as it is recovered. Algorithm CMSHL removes the expression swell by reorganizing the algorithm in [45]. It no longer does any multivariate polynomial multiplications and the algorithm is highly parallelizable.

In this chapter, Sections 4.2.1, 4.3 and 4.4 are new and they are my own contributions. Section 4.2.1 analyzes the expression swell that occurs in Monagan and Tuncer's algorithms which leads to the design and analysis of algorithm CMSHL.

4.1 Steps prior to Hensel lifting

Suppose we seek the factors of a multivariate polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$. Similar to Wang's multivariate Hensel lifting (MHL), a few preliminary steps are performed prior sparse Hensel lifting (SHL) [44, 8]:

The first step is to compute and remove the content of a in a chosen main variable, say x_1. Let $a = \sum_{i=0}^{d_1} a_i(x_2, \cdots, x_n)x_1^i$ where $d_1 = \deg(a, x_1)$. The content of a is $\gcd(a_0, \cdots, a_{d_1})$, a polynomial with one fewer variables which can be factored recursively. Let us assume the content has been removed.

The second step is to identify any repeated factors in a by doing a *square-free factorization* (described in Section 3.4). After this, we obtain the factorization $a = b_1 b_2^2 \cdots b_k^k$ such that each factor b_i is square-free and $\gcd(b_i, b_j) = 1$ for $i \neq j$. Suppose this has also been done and let $a = \mathrm{sqf}(a) = b_1 b_2 \cdots b_k$ be the *square-free part* of a. Let $a = f_1 f_2 \cdots f_r$ be the irreducible factorization over $\mathbb{Z}$. We aim to compute $f_1, \cdots, f_r$.

Next, select a large prime p. For algorithms CMSHL and MTSHL, we assume that p does not divide any term in any irreducible factor f_ρ (for $\rho = 1, \cdots, r$). Then, a positive integer $\tilde{N} < p$ is chosen. Next, an evaluation point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ is chosen randomly from $[1, \tilde{N} - 1]^{n-1}$ and $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}$.

For simplicity, we consider the irreducible factors that are monic in x_1. For the non-monic case, the leading coefficients of the factors can be found by Wang's leading coefficient correction algorithm (LCC) [54]. Wang's LCC works well in practice and Maple currently uses it. If the input polynomial a is represented by a black box, then the non-monic case requires a different approach (see [10] and Section 6.1 for details).

We further restrict to consider only two irreducible factors in this chapter. For MTSHL, the multi-factor case is considered in [44]. For algorithm CMSHL, the two-factor case can be easily extended to multi-factors, and for the black box representation (Section 5.3). Let $a = fg$ where f and g are irreducible polynomials in $\mathbb{Z}[x_1, \cdots, x_n]$ and monic in x_1. We define $h_j := h(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ for a polynomial $h \in \mathbb{Z}[x_1, \cdots, x_n]$.

Now we are ready to start the process of sparse Hensel lifting to recover f and $g \bmod p$ from $a \in \mathbb{Z}[x_1, \cdots, x_n]$, $f_1 = f(x_1, \boldsymbol{\alpha}) \bmod p$, $g_1 = g(x_1, \boldsymbol{\alpha}) \bmod p$. The input and output of sparse Hensel lifting are:

- Input: a prime p, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, $a \in \mathbb{Z}_p[x_1, \cdots, x_n]$, $f_1, g_1 \in \mathbb{Z}_p[x_1]$ s.t.
  (i) $\gcd(f_1, g_1) = 1$ in $\mathbb{Z}_p[x_1]$;
  (ii) $a_1 = f_1 g_1 \in \mathbb{Z}_p[x_1]$;
  (iii) f_1 and g_1 are monic in x_1.

- Output: $f_n, g_n \in \mathbb{Z}_p[x_1, \cdots, x_n]$ s.t.
  (i) $a_n = f_n g_n \in \mathbb{Z}_p[x_1, \cdots, x_n]$;
  (ii) $f_n(x_1, \boldsymbol{\alpha}) = f_1$ and $g_n(x_1, \boldsymbol{\alpha}) = g_1$;

Algorithm 6 MTSHL: Hensel lift x_j with MDPs via sparse interpolation ($j > 2$).

Input: A prime p, $\alpha_j \in \mathbb{Z}$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ both monic in x_1 s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$.

Output: $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ both monic in x_1 s.t.
(i) $a_j = f_j g_j$, (ii) $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Or FAIL.

1: $(\sigma_0, \tau_0) \leftarrow (f_{j-1}, g_{j-1})$; $(f_j, g_j) \leftarrow (f_{j-1}, g_{j-1})$.
2: error $\leftarrow a_j - f_j g_j$. // Computed in $\mathbb{Z}_p[x_1, \cdots, x_n]$
3: monomial $\leftarrow 1$.
4: **for** $i = 1, 2, \cdots$ **while** error $\neq 0$ and $\deg(f_j, x_j) + \deg(g_j, x_j) < \deg(a_j, x_j)$ **do**
5: monomial $\leftarrow$ monomial $\cdot\ (x_j - \alpha_j)$.
6: $c_i \leftarrow$ coeff(error, $(x_j - \alpha_j)^i$).
7: **if** $c_i \neq 0$ **then**
8: Solve the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ for $\sigma_i, \tau_i \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ with $\deg(\sigma_i, x_1) < \deg(f_{j-1}, x_1)$.
9: $\sigma_f \leftarrow \sigma_{i-1}$; $\tau_f \leftarrow \tau_{i-1}$. // Applying the strong SHL
10: $(\sigma_i, \tau_i) \leftarrow$ SparseInterp$(g_{j-1}, f_{j-1}, c_i, \sigma_f, \tau_f)$ // Algorithm 7
11: **if** $(\sigma_i, \tau_i) =$ FAIL **then return** FAIL **end if**
12: $(f_j, g_j) \leftarrow (f_j + \sigma_i \cdot$ monomial$, g_j + \tau_i \cdot$ monomial$)$.
13: error $\leftarrow a_j - f_j g_j$. // Multiplication is in $\mathbb{Z}_p[x_1, \cdots, x_j]$
14: **end if**
15: **end for**
16: **if** error $= 0$ **then return** (f_j, g_j) **else return** FAIL **end if**

(iii) f_n and g_n are monic in x_1.
Or FAIL.

Let $d_{f_j} = \deg(f, x_j)$ and $d_{g_j} = \deg(g, x_j)$. Let

$$f_j = \sum_{i=0}^{d_{f_j}} \sigma_{ji}(x_j - \alpha_j)^i \text{ and } g_j = \sum_{i=0}^{d_{g_j}} \tau_{ji}(x_j - \alpha_j)^i$$

where $\sigma_{ji}, \tau_{ji} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ and $\sigma_{j0} = f_{j-1}$ and $\tau_{j0} = g_{j-1}$. The sparse Hensel lifting algorithm lifts (f_{j-1}, g_{j-1}) to (f_j, g_j) for $2 \leq j \leq n$ by computing the σ_{ji}'s and τ_{ji}'s using sparse interpolation. At each step, $a_j = f_j g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ so that at the final step, $a_n = f_n g_n \in \mathbb{Z}_p[x_1, \cdots, x_n]$.

If the initial evaluation point $\boldsymbol{\alpha}$ is non-Hilbertian, then FAIL is returned and it means no such f_n and g_n exist. If $\boldsymbol{\alpha}$ is Hilbertian, then both algorithms MTSHL (Algorithm 6) and CMSHL (Algorithm 9) can still return FAIL with a low probability (both MTSHL and CMSHL are Las Vegas). Algorithm MTSHL could return FAIL if the sparse interpolation algorithm for computing the MDPs (Algorithm 7) returns FAIL (at step 10 of Algorithm 6). This means the strong SHL assumption failed. Algorithm CMSHL could return FAIL at several steps, i.e. at steps 4, 9, or 19 of Algorithm 9.

Algorithm 7 SparseInterp: solve an MDP using sparse interpolation.

Input: $u, w, c, \sigma_f, \tau_f \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ where u, w are monic in x_1.
Output: The unique solution (σ, τ) to the MDP $\sigma u + \tau w = c \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ with $\deg(\sigma, x_1) < \deg(w, x_1)$ or FAIL.

Let $d\sigma = \deg(\sigma_f, x_1)$ and $\sigma = \sum_{i=0}^{d\sigma} \zeta_i(x_2, \cdots, x_{j-1}) x_1^i$ with $\zeta_i = \sum_{l=1}^{s_i} a_{il} M_{il}$
and $d\tau = \deg(\tau_f, x_1)$ and $\tau = \sum_{i=0}^{d\tau} \eta_i(x_2, \cdots, x_{j-1}) x_1^i$ with $\eta_i = \sum_{l=1}^{t_i} b_{il} N_{il}$, where
a_{il}, b_{il} are to be determined, $x_1^i M_{il}$, $x_1^i N_{il}$ are the monomials in σ_f, τ_f respectively.
Let s be the maximum of the s_i and t_i.
Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
Evaluate monomials at $\boldsymbol{\beta}$: ..... $\mathcal{O}((j-2))(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{il} = M_{il}(\boldsymbol{\beta}) : 1 \le l \le s_i\}, 0 \le i \le d\sigma\}$ and
$\mathcal{T} = \{T_i = \{n_{il} = N_{il}(\boldsymbol{\beta}) : 1 \le l \le t_i\}, 0 \le i \le d\tau\}$.
// Require distinct monomial evaluations to get unique solutions for Vandermonde solves:
if any $|S_i| \neq s_i$ **or** $|T_i| \neq t_i$ **then return** FAIL **end if**
for k from 1 to s **do**
    Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
    Evaluate u, w, c at Y_k. ..... $\mathcal{O}(s(\#f + \#g + \#a))$
    if $\gcd(u(x_1, Y_k), w(x_1, Y_k)) \neq 1$ **then return** FAIL **end if**
    Solve $\sigma_k(x_1)u(x_1, Y_k) + \tau_k(x_1)w(x_1, Y_k) = c(x_1, Y_k) \in \mathbb{Z}_p[x_1]$. ..... $\mathcal{O}(s\, d_1^2)$
end for
for i from 0 to $d\sigma$ **do**
    Construct and solve the $s_i \times s_i$ linear system for the unique solution a_{il}: ... $\mathcal{O}(s\#f)$
    $\left\{\sum_{l=1}^{s_i} a_{il} m_{il}^k = \text{coeff}(\sigma_k(x_1), x_1^i) \text{ for } 1 \le k \le s_i\right\}$.
end for
Substitute the solution a_{il} into σ.
Similarly, construct τ. ..... $\mathcal{O}(s\#g)$
if $\sigma u + \tau w = c$ **then return** (σ, τ) **else return** FAIL **end if** // wrong σ_f or τ_f

To recover the integer coefficients in the factors one must use a sufficiently large p to perform the sparse Hensel lifting (e.g. we use 63-bit primes). For large integer coefficients, a 63-bit prime p may still not be large enough, in that case we could do a subsequent p-adic lift [44].

4.2 Monagan and Tuncer's algorithms

The j^{th} Hensel lifting step for both approaches of Monagan and Tuncer's sparse Hensel lifting algorithms [43, 45] are presented. The first approach (MTSHL [43, 46]) is presented in Algorithms 6 and 7. Algorithm 7 is called from Algorithm 6 in a loop to solve the MDPs via sparse interpolation. Note that in Algorithm 7, if $\max(t_i)$ is much larger (or much smaller) than $\max(s_i)$ then it will be faster to interpolate the smaller of σ and τ only and obtain the larger of σ and τ using $\sigma u + \tau w = c$. The second approach [45] is shown in Algorithm 8. Along with the pseudocode, worst case complexity bounds for the main steps are also included as an aid for the reader and for later reference.

Algorithm 8 Hensel lift x_j via bivariate Hensel lifting ($j > 2$) [45].

Input: A prime p, $\alpha_j \in \mathbb{Z}$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ both monic in x_1 s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$.

Output: $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ both monic in x_1 s.t. (i) $a_j = f_j g_j$, (ii) $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Or FAIL.

1: Let $\sigma_0 = f_{j-1}$, $f_j = \sum_{h=0}^{df_j} \sigma_h(x_1, ..., x_{j-1})(x_j - \alpha_j)^h$ with $\sigma_h = \sum_{i=0}^{df} \left(\sum_{l=1}^{s_i} c_{hil} M_{il}\right) x_1^i$.
Let $\tau_0 = g_{j-1}$, $g_j = \sum_{h=0}^{dg_j} \tau_h(x_1, ..., x_{j-1})(x_j - \alpha_j)^h$ with $\tau_h = \sum_{i=0}^{dg} \left(\sum_{l=1}^{t_i} d_{hil} N_{il}\right) x_1^i$.
$M_{il}x_1^i$, $N_{il}x_1^i$ are monomials in σ_0, τ_0. $df = \deg(f_{j-1}, x_1)$, $dg = \deg(g_{j-1}, x_1)$.
We are to determine: c_{hil}, d_{hil}, $df_j = \deg(f_j, x_j)$, $dg_j = \deg(g_j, x_j)$.
2: Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
3: Evaluate monomials at $\boldsymbol{\beta}$ ..... $\mathcal{O}((j-2)(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{il} = M_{il}(\boldsymbol{\beta}), 1 \le l \le s_i\}, 0 \le i \le df - 1\}$ and
$\mathcal{T} = \{T_i = \{n_{il} = N_{il}(\boldsymbol{\beta}), 1 \le l \le t_i\}, 0 \le i \le dg - 1\}$.
4: **if** any $|\mathsf{S_i}| \neq \mathsf{s_i}$ **or** any $|\mathsf{T_i}| \neq \mathsf{t_i}$ **then return** FAIL **end if** // Require distinct monomial evaluations so that the solutions for the Vandermonde solves are unique.
5: Let s be the maximum of the s_i and t_i.
6: **for** k from 1 to s **do**
7: Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
8: $A_k, F_k, G_k \leftarrow a_j(x_1, Y_k, x_j), f_{j-1}(x_1, Y_k), g_{j-1}(x_1, Y_k)$. ..... $\mathcal{O}(s(\#f + \#g + \#a))$
9: **if** $\gcd(F_k, G_k) \neq 1$ **then return** FAIL **end if** // unlucky evaluation
10: Call BivariateHenselLift$(A_k, F_k, G_k, \alpha_j, p)$ to compute $\sigma_{hk}(x_1)$ and $\tau_{hk}(x_1)$ s.t. $A_k = f_k g_k$ where $f_k = \sum_{h=0}^{df_j} \sigma_{hk}(x_j - \alpha_j)^h$ and $g_k = \sum_{h=0}^{dg_j} \tau_{hk}(x_j - \alpha_j)^h$.
11: **end for**
12: **for** h from 1 to df_j **do**
13: **for** i from 0 to df **do**
14: Construct and solve the $s_i \times s_i$ linear system for the unique solution c_{hil}: $\mathcal{O}(sd_j\#f)$ $\left\{\sum_{l=1}^{s_i} c_{hil} m_{il}^k = \text{coeff}(\sigma_{hk}(x_1), x_1^i) \text{ for } 1 \le k \le s_i\right\}$
15: **end for**
16: **end for**
17: Substitute the c_{hil}'s into $\sigma_h = \sum_{i=0}^{df} \left(\sum_{l=1}^{s_i} c_{hil} M_{il}\right) x_1^i$ for $0 \le h \le df_j$ and expand $\sum_{h=0}^{df_j} \sigma_h(x_1, ..., x_{j-1})(x_j - \alpha_j)^h$ to get f_j. ..... $\mathcal{O}(d_j^2\#f)$
18: Similarly to construct g_j. ..... $\mathcal{O}(sd_j\#g)$
19: **if** $a_j = f_j g_j$ **then return** (f_j, g_j) **else return** FAIL **end if** ..... $\mathcal{O}(\#f\#g)$

4.2.1 Intermediate expression swell

Definition 4.2.1. An **intermediate expression swell** occurs if an intermediate polynomial has more terms than both the input and output polynomials.

The presence of an intermediate expression swell makes it harder to bound the complexity of an algorithm. In Algorithm 6, an expression swell may occur in steps 12 and 13. In Algorithm 8 an expression swell may occur at the final expansion step (step 17). To study the expression swell, we consider the partial sums of f_j, $f_j^{(i)}$ and $f_{jH}^{(i)}$, which correspond to

step 12 of Algorithm 6 and in step 17 of Algorithm 8 respectively. Let

$$f_j^{(i)} = \sum_{k=0}^{i} \sigma_k(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^k \text{ for } 0 \le i \le d_{f_j} \text{ and}$$
$$f_{jH}^{(i)} = (((\sigma_{d_{f_j}}(x_j - \alpha_j) + \sigma_{d_j-1})(x_j - \alpha_j) + \cdots)(x_j - \alpha_j)) + \sigma_{d_j-i}$$

where $f_{jH}^{(i)}$ is the expansion using Horner's form and $d_{f_j} = \deg(f_j, x_j)$. We shall look at $\#f_j^{(i)}$ and $\#f_{jH}^{(i)}$ for all $0 \le i \le d_{f_j}$.

Table 4.1 shows an example of a randomly generated polynomial with $p = 2^{31} - 1$, $j = 5$, $d_{f_j} = 14$, $d = \deg(a) = 20$ and $\#f_j = 989$. The density ratio $\#f_j / \binom{d+j}{j} \approx 0.0186$. The ratios $\max\left(\#f_j^{(i)}\right) / \#f_j$ and $\max\left(\#f_{jH}^{(i)}\right) / \#f_j$ are $2021/989 = 2.043$ and $2983/989 = 3.016$ respectively. This example shows a typical trend in an average case where $\max\left(\#f_j^{(i)}\right) / \#f_j \lessapprox 1 + d/j$ [46]. We observe that $\#\sigma_i$ decreases as i increases from 0 to d_{f_j}. The number of terms $\#f_j^{(i)}$ increases to a peak in the first few expansions and gradually shrinks back to $\#f_j$, whereas $\#f_{jH}^{(i)}$ increases to a higher peak than $\max(\#f_j^{(i)})$ and drops down to $\#f_j$ at the last iteration.

i	0	1	2	3	4	5	6	7
$\#\sigma_i$	925	737	584	459	352	268	196	134
$\#f_j^{(i)}$	925	1512	1851	1999	**2021**	1934	1768	1628
$\#f_{jH}^{(i)}$	3	10	23	47	95	159	253	387
i	8	9	10	11	12	13	14	max
$\#\sigma_i$	94	64	48	24	13	7	3	-
$\#f_j^{(i)}$	1486	1411	1226	1130	1071	1028	989	**2021**
$\#f_{jH}^{(i)}$	583	851	1203	1662	2246	**2983**	989	**2983**

Table 4.1: Number of terms in $f_j^{(i)}$ and $f_{jH}^{(i)}$ with a randomly generated polynomial.

i	0	1	2	3	4
$\#\sigma_i$	7	7	7	7	7
$\#f_j^{(i)}$	7	14	21	**28**	7
$\#f_{jH}^{(i)}$	7	14	21	**28**	7

Table 4.2: Number of terms in $f_j^{(i)}$ and $f_{jH}^{(i)}$ in a worst case.

The following example illustrates the worst case when both $\#f_j^{(i)}$ and $\#f_{jH}^{(i)}$ increase linearly to their maximum, $d_{f_j} \cdot (\#f_j)$.

Example 12. Let $p = 101$, $j = 4$,

$$f_j = (31x_3 + 100x_3^2 + (49 + 36x_2^2 + (x_1^4 + 44x_1^2 + 28)x_2^3)x_3^3)x_4^4$$

with $\#f_j = 7$ and $d_{f_j} = 4$. Table 4.2 shows the number of terms in $f_j^{(i)}$ and $f_{jH}^{(i)}$. Here, $\max\left(\#f_j^{(i)}\right) = \max\left(\#f_{jH}^{(i)}\right) = d_{f_j} \cdot (\#f_j) = 4 \cdot 7 = 28$. Since $\sigma_i = \frac{1}{i!}\frac{\partial^{(i)} f_j}{\partial x_j^i}(x_j = \alpha_j)$ for $0 \leq i \leq d_{f_j}$ and f_j only contains terms with x_j^4, $\#\sigma_i$ remains constant as i increases from 0 to d_{f_j}. Thus, we have $\max\left(\#f_j^{(i)}\right)/\#f_j = d_{f_j}$.

4.3 Our new algorithm CMSHL

We present a new algorithm which eliminates the expression swell in Algorithm 8. The idea is depicted in Figure 4.1.

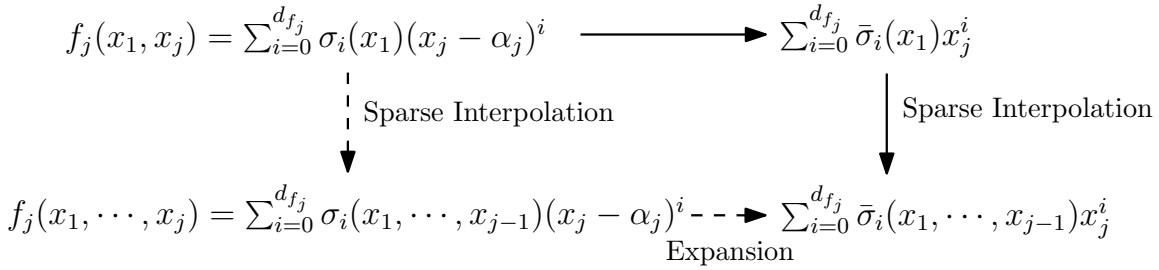

Figure 4.1: Dashed arrow: Algorithm 8 [45], expression swell occurs at the expansion step. Lined arrow: CMSHL (Algorithm 9).

Consider one of the factors f_j at the j^{th} Hensel lifting step:

$$f_j(x_1, \cdots, x_j) = \sum_{i=0}^{d_{f_j}} \sigma_i(x_1, \cdots, x_{j-1})(x_j - \alpha_j)^i = \sum_{i=0}^{d_{f_j}} \bar{\sigma}_i(x_1, \cdots, x_{j-1}) x_j^i, \tag{4.1}$$

where $d_{f_j} = \deg(f_j, x_j)$. There are two routes to recover $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ in (4.1) from its bivariate image $f_j(x_1, x_j)$. One route is to first recover $\sigma_i(x_1, \cdots, x_{j-1})$ from $\sigma_i(x_1)$ using sparse interpolation and then expand to get $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ in (4.1) (through the dashed arrows in Figure 4.1). This was done by Monagan and Tuncer in [45] (Algorithm 8). In our new algorithm (CMSHL), bivariate images are computed and expanded first and then the coefficients $\bar{\sigma}_i(x_1, \cdots, x_{j-1})$ are recovered directly from $\bar{\sigma}_i(x_1)$ to get the final expanded form. This is through the lined arrows in Figure 4.1. Multivariate polynomial expansions in step 13 of Algorithm 6 and step 17 of Algorithm 8 are avoided where expression swells can occur.

Our new algorithm is presented in Algorithm 9 and we call it **CMSHL** (Chen-Monagan sparse Hensel lifting). The correctness of this algorithm is based on the following: since all loop ranges are finite, algorithm CMSHL terminates. When algorithm CMSHL terminates, it outputs either two factors f_n, g_n or FAIL. Since Algorithm 9 tests if $a_j = f_j g_j$ in step 19,

Algorithm 9 CMSHL: Hensel lifting x_j via bivariate Hensel lifting ($j > 2$).

Input: A prime p, $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ monic in x_1,
$f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ both monic in x_1 s.t. $a_j(x_1, ..., x_{j-1}, \alpha_j) = f_{j-1}g_{j-1}$.
Output: $f_j, g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$ both monic in x_1 s.t.
(i) $a_j = f_j g_j$, (ii) $f_j(x_j = \alpha_j) = f_{j-1}$ and $g_j(x_j = \alpha_j) = g_{j-1}$; Or FAIL.

Let $f_{j-1} = x_1^{df} + \sum_{i=0}^{df-1} \sigma_i(x_2, ..., x_{j-1}) x_1^i$ with $\sigma_i = \sum_{k=1}^{s_i} c_{ik} M_{ik}$
and $g_{j-1} = x_1^{dg} + \sum_{i=0}^{dg-1} \tau_i(x_2, ..., x_{j-1}) x_1^i$ with $\tau_i = \sum_{k=1}^{t_i} d_{ik} N_{ik}$,
where M_{ik}, N_{ik} are the monomials in σ_i, τ_i respectively.
Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
Evaluate monomials at $\boldsymbol{\beta}$: ..... $\mathcal{O}((j-2)(\#f + \#g + d_{\max}))$
$\mathcal{S} = \{S_i = \{m_{ik} = M_{ik}(\boldsymbol{\beta}), 1 \le k \le s_i\}, 0 \le i \le df - 1\}$ and
$\mathcal{T} = \{T_i = \{n_{ik} = N_{ik}(\boldsymbol{\beta}), 1 \le k \le t_i\}, 0 \le i \le dg - 1\}$.
if any $|S_i| \neq s_i$ **or** any $|T_i| \neq t_i$ **then return** FAIL **end if**
Let s be the maximum of the s_i and t_i.
for k from 1 to s **in parallel do**
    Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
    $A_k, F_k, G_k \leftarrow a_j(x_1, Y_k, x_j), f_{j-1}(x_1, Y_k), g_{j-1}(x_1, Y_k)$. ..... $\mathcal{O}(s(\#f + \#g + \#a))$
    if $\gcd(F_k, G_k) \neq 1$ **then return** FAIL **end if** // unlucky evaluation
    $f_k, g_k \leftarrow$ BivariateHenselLift$(A_k, F_k, G_k, \alpha_j, p)$. ..... $\mathcal{O}(s(d_1^2 d_j + d_1 d_j^2))$
end for
Let $f_k = x_1^{df} + \sum_{l=1}^{\mu} \alpha_{kl} \tilde{M}_l(x_1, x_j)$ for $1 \le k \le s$, where $\mu \le d_1 d_j$.
for l from 1 to μ **in parallel do**
    $i \leftarrow \deg(\tilde{M}_l, x_1)$.
    Solve the $s_i \times s_i$ linear system for c_{lk}: $\{\sum_{k=1}^{s_i} m_{ik}^t c_{lk} = \alpha_{nl}$ for $1 \le t \le s_i\}$ $\mathcal{O}(s d_j \#f)$
end for
Construct $f_j \leftarrow x_1^{df} + \sum_{l=1}^{\mu} \left(\sum_{k=1}^{s_i} c_{lk} M_{ik}(x_2, ..., x_{j-1})\right) \tilde{M}_l(x_1, x_j)$.
Similarly, construct g_j. ..... $\mathcal{O}(s d_j \#g)$
if $a_j = f_j g_j$ **then return** (f_j, g_j) **else return** FAIL **end if** ..... $\mathcal{O}(\#f \#g)$

the output (f_j, g_j) is the correct factorization. We present an upper bound on the probability that Algorithm 9 outputs FAIL in Section 4.4.

Algorithm CMSHL also has a significant advantage for parallelization, however, it only uses the **weak SHL assumption** during a sparse interpolation. It can not use the **strong SHL assumption** as in MTSHL. The strong and the weak SHL assumptions are both defined in Section 3.3.

4.4 Complexity analyses with failure probabilities

For both MTSHL and CMSHL, the number of arithmetic operations in $\mathbb{Z}_p$ is bounded for the worst-case, along with their failure probabilities.

4.4.1 MTSHL

MTSHL uses the **strong SHL assumption** to solve the multivariate polynomial Diophantine equations (MDPs) in a loop. The following lemma was proved in [43]:

Lemma 4.4.1. *Let $f \in \mathbb{Z}_p[x_1, \cdots, x_n]$ and let α be a randomly chosen element in $\mathbb{Z}_p$. Let $f = \sum_{i=0}^{d_n} \sigma_i(x_1, \cdots, x_{n-1})(x_n - \alpha)^i$ where $d_n = \deg(f, x_n)$. Then,*

$$\Pr[\mathrm{Supp}(\sigma_{i+1}) \nsubseteq \mathrm{Supp}(\sigma_i)] \leq |\mathrm{Supp}(\sigma_{i+1})| \frac{d_n - i}{p - d_n + i + 1} \text{ for } 0 \leq i < d_n.$$

Step 10 of Algorithm 6 applies the strong SHL assumption by using $\mathrm{Supp}(\sigma_f) = \mathrm{Supp}(\sigma_{i-1})$ and $\mathrm{Supp}(\tau_f) = \mathrm{Supp}(\tau_{i-1})$ as the supports for σ_i and τ_i. Therefore we only need to solve systems of linear equations for the coefficients. This is the key idea used to solve the MDPs in Algorithm 7, which we shall analyze in the following.

The failure probability of the MDPs

There are two places where Algorithm 7 can return FAIL intermediately: step 5 and step 9. The failure probabilities are bounded as follows. The proof closely follows Monagan and Tuncer's method in [46], but I do it for the worst-case analysis instead of the average-case as in [46]. I present the proof below since I use their method in the failure probability analysis of CMSHL.

In order to prove Proposition 4.4.4, we need Definition 4.4.2 (in Chapter 3 of [14]) and Lemma 4.4.3 (Lemma 4 in [24]).

Definition 4.4.2. Let R be a commutative ring and let $A, B \in R[x_1]$. Let $A = \sum_{i=0}^{s} a_i x_1^i$ and $B = \sum_{j=0}^{t} b_j x_1^j$ with $s > 0$ and $t > 0$, where $a_i, b_j \in R$. The **Sylvester matrix** of A and B with respect to x_1, denoted as $\mathrm{Syl}(A, B, x_1)$, is the following $s + t$ by $s + t$ matrix

$$\mathrm{Syl}(A, B, x_1) = \begin{pmatrix} a_s & 0 & & 0 & b_t & 0 & & 0 \\ a_{s-1} & a_s & & 0 & b_{t-1} & b_t & & 0 \\ \vdots & a_{s-1} & \ddots & 0 & \vdots & b_{t-1} & \ddots & 0 \\ a_1 & \vdots & & a_s & b_1 & \vdots & & b_t \\ a_0 & a_1 & & a_{s-1} & b_0 & b_1 & & b_{t-1} \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ 0 & 0 & \ddots & a_1 & 0 & 0 & \ddots & b_1 \\ 0 & 0 & & a_0 & 0 & 0 & & b_0 \end{pmatrix}. \tag{4.2}$$

The **Sylvester resultant** of A and B with respect to x_1, denoted as $\mathrm{res}(A, B, x_1)$, is the determinant of the Sylvester matrix $\mathrm{Syl}(A, B, x_1)$.

A key property of the resultant is that if $R = \mathbb{F}$ a field then

$$\deg(\gcd(A, B), x_1) > 0 \iff \operatorname{res}(A, B, x_1) = 0. \tag{4.3}$$

Lemma 4.4.3. *Let $\mathbb{F}$ be a field and let $A, B \in \mathbb{F}[x_2, \cdots, x_n][x_1]$ with $\deg(A, x_1) > 0$ and $\deg(B, x_1) > 0$. Let $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_n) \in \mathbb{F}^{n-1}$. If $\mathrm{LC}(A)(\boldsymbol{\beta}) \neq 0$ and $\mathrm{LC}(B)(\boldsymbol{\beta}) \neq 0$, then*

$$\deg(\gcd(A(x_1, \boldsymbol{\beta}), B(x_1, \boldsymbol{\beta})), x_1) > 0 \iff \operatorname{res}(A(x_1, \boldsymbol{\beta}), B(x_1, \boldsymbol{\beta}), x_1) = 0.$$

Proposition 4.4.4. *Let p be a large prime, $d = \deg(a)$ and s be the integer defined in step 2 of Algorithm 7. When Algorithm 6 calls Algorithm 7 with inputs $(u, w, c, \sigma_f, \tau_f) = (g_{j-1}, f_{j-1}, c_i, \sigma_{i-1}, \tau_{i-1})$, if $\mathrm{Supp}(\sigma_i) \subseteq \mathrm{Supp}(\sigma_{i-1})$ and $\mathrm{Supp}(\tau_i) \subseteq \mathrm{Supp}(\tau_{i-1})$, for $i = 1, 2, 3, \cdots$, then Algorithm 7 fails to compute (σ_i, τ_i) for the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ with a probability less than*

$$\underbrace{\frac{d\, s(\#f_{j-1} + \#g_{j-1})}{2(p-1)}}_{\text{step 5}} + \underbrace{\frac{d^2 s^2}{p-1}}_{\text{step 9}}. \tag{4.4}$$

Proof. For step 5, let $\Delta_i = \prod_{1 \le l < k \le s_i} (M_{il} - M_{ik})$, where M_{il}, M_{ik} are monomials in $\mathcal{S}$ defined in step 4. Let $\Delta = \prod_{i=0}^{d\sigma} \Delta_i$, where $d\sigma = \deg(\sigma_f, x_1)$. Then $\Delta(\boldsymbol{\beta}) = 0$ implies $\Delta_i(\boldsymbol{\beta}) = 0$ for some i so that not all monomial evaluations are distinct. Also, $\deg(M_{il}) \le d$ for each monomial in $\mathcal{S}$ and $\sum_{i=0}^{d\sigma} s_i = \#f_{j-1}$. Thus,

$$\deg(\Delta) \le \sum_{i=0}^{d\sigma} d \binom{s_i}{2} \le \frac{d\, s}{2} \sum_{i=0}^{d\sigma} (s_i - 1) < \frac{d\, s \# f_{j-1}}{2}.$$

By the Schwartz-Zippel lemma (Lemma 1.6.4), since the $\beta_i' s$ are chosen randomly from $[1, p-1]$,

$$\Pr[\Delta(\boldsymbol{\beta}) = 0] \le \frac{\deg(\Delta)}{p-1} < \frac{d\, s \# f_{j-1}}{2(p-1)}.$$

Similarly, the monomial evaluations for τ are considered. Thus,

$$\Pr[\text{Algorithm 7 fails at step 5}] < \frac{d\, s(\#f_{j-1} + \#g_{j-1})}{2(p-1)}.$$

To be able to solve the Diophantine equation in step 10, we need

$$\gcd(u(x_1, Y_k), w(x_1, Y_k)) = \gcd(g_{j-1}(x_1, Y_k), f_{j-1}(x_1, Y_k)) = 1.$$

Let $R = \mathrm{res}(g_{j-1}, f_{j-1}, x_1) \in \mathbb{Z}_p[x_2, \cdots, x_{j-1}]$. Since f_{j-1} and g_{j-1} are monic in x_1, by Lemma 4.4.3, step 9 returns FAIL if

$$\deg(\gcd(g_{j-1}(x_1, Y_k), f_{j-1}(x_1, Y_k)), x_1) > 0 \iff R(Y_k) = 0 \iff R(\beta_2^k, \cdots, \beta_{j-1}^k) = 0.$$

Let $S = \prod_{k=1}^{s} R(x_2^k, x_3^k, \cdots, x_{j-1}^k)$. Since $\deg(f_{j-1}) < d$ and $\deg(g_{j-1}) < d$, $\deg(R) < d^2$ (from Definition 4.4.2) and

$$\deg(S) = \sum_{k=1}^{s} k \deg(R) < \sum_{k=1}^{s} kd^2 = \frac{d^2 s(s+1)}{2} \leq d^2 s^2.$$

By Lemma 1.6.4, since $\beta_i' s$ are chosen randomly from $[1, p-1]$,

$$\Pr[R(Y_k) = 0 \text{ for some } k] = \Pr[S(\boldsymbol{\beta}) = 0] \leq \frac{\deg(S)}{p-1} < \frac{d^2 s^2}{p-1}.$$

Adding the failure probabilities at step 5 and 9, we obtain the result. □

At the end of Algorithm 7, at step 17, $\sigma u + \tau w = c$ can be checked probabilistically with a single evaluation point. If Algorithm 7 returns FAIL at step 17, the support in either σ_f or τ_f was wrong (the strong SHL assumption fails). By Lemma 4.4.1, Algorithm 6 fails at the j^{th} Hensel lifting step due to a wrong support in σ_i with a probability no more than

$$\sum_{i=0}^{d_j - 1} |\mathrm{supp}(\sigma_{i+1})| \frac{d_j - i}{p - d_j + i + 1} \leq \#f_{j-1} \sum_{i=0}^{d_j - 1} \frac{d_j - i}{p - d_j + i + 1} < \frac{d_j(d_j+1)\#f_{j-1}}{2(p - d_j + 1)},$$

where $d_j = \deg(a, x_j)$.

Note that the number s in Proposition 4.4.4 varies since MDPs are called in a loop from Algorithm 6. We denote $s_{j,i}$ as the maximum number of monomials in the coefficients of σ_{i-1} and τ_{i-1} in x_1 for the i^{th} call of the MDP in the j^{th} Hensel lifting step. Let $s_j = \max_i(s_{j,i})$ and $T_{j-1} = \#f_{j-1} + \#g_{j-1}$. We have $d_j \leq d$. We assume that $\boldsymbol{\alpha}$ is Hilbertian. Adding up the failure probabilities at step 5, 9 and 17, we obtain the failure probability at the j^{th} Hensel lifting step:

Proposition 4.4.5. *Let p be a large prime, $d = \deg(a)$, $s_j = \max_i(s_{j,i})$ and $T_{j-1} = \#f_{j-1} + \#g_{j-1}$. Assume that $\boldsymbol{\alpha}$ is Hilbertian. Algorithm 6 (MTSHL) fails to compute f_j and g_j from f_{j-1} and g_{j-1} at the j^{th} Hensel lifting step ($j > 2$) by Algorithm 7 with a probability less than*

$$\frac{d^2(2s_j^2 + T_{j-1}) + d(s_j + 1)T_{j-1}}{2(p - d + 1)}. \tag{4.5}$$

For the whole MTSHL process (for $2 \leq j \leq n$), we have $\#f_{j-1} \leq \#f$, $\#g_{j-1} \leq \#g$.

Proposition 4.4.6. *Let p be a large prime s.t. p does not divide any term of f and g, n be the number of variables in a, $d = \deg(a)$, $s_{\max} = \max(s_j)$ and $T_{fg} = \#f + \#g$. Assume that $\boldsymbol{\alpha}$ is Hilbertian. MTSHL (the j^{th} Hensel lifting step as in Algorithm 6) fails to solve the MDP via sparse interpolation (Algorithm 7) with a probability less than*

$$\frac{(n-2)(d^2(2s_{\max}^2 + T_{fg}) + d(s_{\max}+1)T_{fg})}{2(p-d+1)}. \tag{4.6}$$

We illustrate the probability in Proposition 4.4.6 for a typical large factorization problem. Let $n = 10$, $d = 10^2$, $T_{fg} = 2 \times 10^4$ and $s_{\max} = 10^2$. If $p = 2^{63} - 25$, then MTSHL fails with probability less than 2.611×10^{-10}.

The complexity of solving the MDP

It remains to bound the number of arithmetic operations in $\mathbb{Z}_p$ for Algorithm 7. Theorem 4.4.7 gives the complexity of solving the MDP.

Theorem 4.4.7. *Let p be a large prime, s be the integer defined in step 2 of Algorithm 7, $d_1 = \deg(a, x_1)$ and a_j $(j > 2)$ be monic in x_1. We assume $j - 2 \lessapprox s$. Let $T_{j-1} = \#f_{j-1} + \#g_{j-1}$. When Algorithm 6 calls Algorithm 7, if Algorithm 7 succeeds (the failure probability is bounded by (4.5)), the number of arithmetic operations in $\mathbb{Z}_p$ for solving the MDP $\sigma_i g_{j-1} + \tau_i f_{j-1} = c_i$ for $i = 1, 2, \cdots$ using Algorithm 7 in the worst case is $\mathcal{O}(s(\#a_j + T_{j-1} + d_1^2))$.*

Proof. Let $d_{\max} = \max_{i=2}^{j-1} \deg(a, x_i)$ and $d_{f_{\max}} = \max_{i=2}^{j-1} \deg(f, x_i)$.

For step 4 of Algorithm 7, one way to evaluate the monomials is to create a table of powers for each variable $x_2, \cdots, x_{j-1}$, as shown in Fig. 4.2. Let $d\sigma_{f_i} = \deg(\sigma_f, x_i)$. It takes $\sum_{i=2}^{j-1}(d\sigma_{f_i} - 1) \leq (j-2)(d_{f_{\max}} - 1)$ multiplications to compute the table. After creating the table, it takes $\mathcal{O}\left((j-3)\sum_{i=0}^{d\sigma} s_i\right) = \mathcal{O}((j-3)\#\sigma_f)$ multiplications to evaluate monomials in $\mathcal{S}$. Similarly for the evaluations in $\mathcal{T}$. Thus, the total cost is $\mathcal{O}((j-2)(\#\sigma_f + \#\tau_f + d_{\max}))$ arithmetic operations in $\mathbb{Z}_p$.

1	β_2	β_2^2	$\cdots$	$\beta_2^{d\sigma_{f_2}}$
1	β_3	β_3^2	$\cdots$	$\beta_3^{d\sigma_{f_3}}$
$\vdots$				
1	β_{j-1}	β_{j-1}^2	$\cdots$	$\beta_{j-1}^{d\sigma_{f_{j-1}}}$

Figure 4.2: Evaluation table for variables $x_2, \cdots, x_{j-1}$.

For step 8 of Algorithm 7, the monomial evaluations and the coefficients of u are stored in two arrays (M and C) of each size $\#u$. Similarly for w and c. At the first iteration, each entry in M is squared and then multiplied by the corresponding coefficient in C to compute the sum. Each iteration costs $3(\#u+\#w+\#c)$ arithmetic operations in $\mathbb{Z}_p$. The total cost is $\mathcal{O}(s(\#f_{j-1}+\#g_{j-1}+\#a_j))$.

For step 10, solving each univariate Diophantine equation using the extended Euclidean algorithm costs $\mathcal{O}(d_1^2)$ arithmetic operations in $\mathbb{Z}_p$ (for a reference, see Chapter 2 of [19]).

For steps 12 to 14, the Vandermonde solver (Zippel's algorithm [62]) costs $\sum_{i=0}^{d\sigma}\mathcal{O}(s_i^2)\subseteq\mathcal{O}(s\#\sigma_f)$ arithmetic operations in $\mathbb{Z}_p$.

We have $\#\sigma_f\le\#f_{j-1}$ and $\#\tau_f\le\#g_{j-1}$. The total cost of Algorithm 7 is

$$\begin{aligned}&\mathcal{O}(\underbrace{s(\#f_{j-1}+\#g_{j-1}+\#a_j)}_{\text{eval in step 8}}+\underbrace{s\,d_1^2}_{\text{step 10}}+\underbrace{s(\#\sigma_f+\#\tau_f)}_{\text{VS in steps 12--14}})\\ \subseteq\ &\mathcal{O}(s(\#a_j+\#f_{j-1}+\#g_{j-1}+d_1^2))=\mathcal{O}(s(\#a_j+T_{j-1}+d_1^2))\end{aligned}$$

arithmetic operations in $\mathbb{Z}_p$ in the worst case. □

The complexity of MTSHL

Now we return to the analysis of Algorithm 6 – Hensel lifting x_j with multivariate polynomial Diophantine equations. One bottleneck of Algorithm 6 is the error computation at step 13. There is an expression swell of f_j and g_j at step 12 of up to a factor of $d_j=\deg(a,x_j)$ (Example 12). Theorem 4.4.8 gives the complexity at the j^{th} Hensel lifting step.

Theorem 4.4.8. *Let p be a large prime s.t. p does not divide any term of f and g. Let $d_1=\deg(a,x_1)$, $d_j=\deg(a,x_j)$ and $s_j=\max_i(s_{j,i})$. We assume $\boldsymbol{\alpha}$ is Hilbertian. Let $T_{j-1}=\#f_{j-1}+\#g_{j-1}$. If the j^{th} Hensel lifting step of MTSHL (Algorithm 6) succeeds (the failure probability is bounded by (4.5)), the number of arithmetic operations in $\mathbb{Z}_p$ is*

$$\mathcal{O}(\underbrace{d_j^2\#a_j}_{\text{step 6,12}}+\underbrace{d_js_j(\#a_j+T_{j-1}+d_1^2)}_{\text{MDP}}+\underbrace{d_j^3\#f_{j-1}\#g_{j-1}}_{\text{error comp.}}) \tag{4.7}$$

in the worst case.

Proof. To compute coeff(error, $(x_j-\alpha_j)^i$) in step 6, using repeated differentiation and evaluation costs $O(i\#\text{error})$. The total cost is $\mathcal{O}(d_j^2\#a_j)$.

The total cost of solving the MDPs using sparse interpolation in step 10 is $\mathcal{O}(d_js_j(\#a_j+T_{j-1}+d_1^2))$, from Theorem 4.4.7.

The total cost of the error computation in step 13 in the worst case is

$$\mathcal{O}\left(\sum_{k=1}^{d_j}\#f_j^{(k)}\#g_j^{(k)}\right)=\mathcal{O}\left(\sum_{i=1}^{d_j}(i\#f_{j-1})(i\#g_{j-1})\right)\subseteq\mathcal{O}(d_j^3\#f_{j-1}\#g_{j-1})$$

arithmetic operations in $\mathbb{Z}_p$ assuming classical polynomial multiplication.

The total cost for Algorithm 6 is

$$\mathcal{O}(\underbrace{d_j^2 \#a_j}_{\text{step 6,12}} + \underbrace{d_j s_j (\#a_j + T_{j-1} + d_1^2)}_{\text{MDP}} + \underbrace{d_j^3 \#f_{j-1} \#g_{j-1}}_{\text{error comp.}})$$

arithmetic operations in $\mathbb{Z}_p$ in the worst case. □

We remark that in Theorem 4.4.8, the expression swell appears as the factor of d_j^2. On average the expression swell is much less. The complexity of the whole MTSHL process is given by Theorem 4.4.9.

Theorem 4.4.9. *Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ be monic in x_1. Let f and g be the monic irreducible factors of a. Let p be a large prime s.t. p does not divide any term of f and g. Let $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ be a random evaluation point selected from $[1, \tilde{N} - 1]^{n-1}$. Assume $\boldsymbol{\alpha}$ is Hilbertian. Let $T_{fg} = \#f + \#g$. Let $f_1 = f(x_1, \boldsymbol{\alpha})$ and $g_1 = g(x_1, \boldsymbol{\alpha})$ be the image polynomials with $\gcd(f_1, g_1) = 1$. Let $d_i = \deg(a, x_i)$ for $1 \leq i \leq n$, $d_{\max} = \max_{i=2}^{n}(d_i)$ and $s_{\max} = \max_{j=3}^{n}(s_j)$. If Algorithm MTSHL succeeds (the failure probability is bounded by (4.6)), the total number of arithmetic operations in $\mathbb{Z}_p$ for lifting f_1 and g_1 to f_n and g_n in $n-1$ steps in the worst case is*

$$\mathcal{O}(\underbrace{d_1^2 d_2 + d_1 d_2^2}_{\text{first BHL}} + (n-2)(\underbrace{d_{\max}^2 \#a + s_{\max} d_{\max}(\#a + T_{fg} + d_1^2) + d_{\max}^3 \#f \#g}_{\text{MTSHL } x_3, x_4, \cdots, x_n})). \tag{4.8}$$

4.4.2 CMSHL

In algorithm CMSHL, the **weak SHL assumption** is used instead of the **strong SHL assumption** as in MTSHL (see Definition 3.3).

The failure probability of CMSHL

For the j^{th} Hensel lifting step, by Lemma 3.3.2, the failure probability due to a wrong support in either f_j or g_j (Algorithm 9 fails at step 19) is bounded by

$$(\#f_{j-1} + \#g_{j-1}) \sum_{i=1}^{d_j} \frac{d_j}{p - d_j + i} \leq \frac{d_j^2 (\#f_{j-1} + \#g_{j-1})}{p - d_j + 1}.$$

The number s defined in step 5 of Algorithm 9 is equivalent to $s_j = \max(s_{j,i})$ in MTSHL. We denote s_j as the number s in step 5 of Algorithm 9 at the j^{th} Hensel lifting step. Identical to MTSHL (Proposition 4.4.4), the failure probabilities at steps 4 and 9 are

$$\underbrace{\frac{d\, s_j (\#f_{j-1} + \#g_{j-1})}{2(p-1)}}_{\text{step 4}} + \underbrace{\frac{d^2 s_j^2}{p-1}}_{\text{step 9}}.$$

We assume that $\boldsymbol{\alpha}$ is Hilbertian. Adding the failure probabilities at steps 4, 9 and 19, we have the failure probability at the j^{th} Hensel lifting step for Algorithm 9 (CMSHL) is:

Proposition 4.4.10. *Let p be a large prime s.t. p does not divide any term of f and g. Let $d = \deg(a)$, $T_{j-1} = \#f_{j-1} + \#g_{j-1}$ and s_j be the number s defined in step 5 of Algorithm 9 at the j^{th} Hensel lifting step. Then Algorithm 9 fails to compute f_j and g_j from f_{j-1} and g_{j-1} at the j^{th} Hensel lifting step ($j > 2$) with a probability less than*

$$\frac{2d^2(s_j^2 + T_{j-1}) + ds_jT_{j-1}}{2(p - d + 1)}. \tag{4.9}$$

The complexity of CMSHL

Theorem 4.4.11. *Let p be a large prime s.t. p does not divide any term of f and g. Let $d_j = \deg(a, x_j)$ for $1 \le j \le n$ and s_j be the number s defined in step 5 of Algorithm 9 for the j^{th} Hensel lifting step. Let $T_{j-1} = \#f_{j-1} + \#g_{j-1}$. If Algorithm 9 succeeds (the failure probability is bounded by (4.9)), the number of arithmetic operations in $\mathbb{Z}_p$ for the j^{th} Hensel lifting step of CMSHL in the worst case is*

$$\mathcal{O}(d_js_j(T_{j-1} + d_1^2 + d_1d_j) + s_j\#a_j + \#f_j\#g_j). \tag{4.10}$$

Proof. Similar to the analysis of Algorithm 7, we bound the total number of arithmetic operations in $\mathbb{Z}_p$ for the worst case in Algorithm 9. Let $d_{\max} = \max_{i=2}^{j-1} d_i$.

The total cost of evaluations in step 3 is $\mathcal{O}((j-2)(\#f_{j-1} + \#g_{j-1} + d_{\max}))$.

The total cost of step 8 is $\mathcal{O}(s_j(\#f_{j-1} + \#g_{j-1} + \#a_j))$.

Each bivariate Hensel lift in line 10 costs $\Theta(d_1d_j^2 + d_jd_1^2)$ [39, 48].

Using Zippel's algorithm [62], the total cost of the Vandermonde solver in step 15 is $d_j \sum_{i=0}^{df-1} \Theta(s_i^2) \subseteq \mathcal{O}(d_js_j\#f_{j-1})$ for f_j. Similarly, for g_j, we have $\mathcal{O}(d_js_j\#g_{j-1})$.

Finally, in step 19, the cost of multiplying f_j and g_j is $\mathcal{O}(\#f_j\#g_j)$.

Assuming $j - 2 \lessapprox s_j$, the total cost of Algorithm 9 is

$$\mathcal{O}(\underbrace{s_j(\#f_{j-1} + \#g_{j-1} + \#a_j)}_{\text{eval in step 8}} + \underbrace{s_j(d_1^2d_j + d_1d_j^2)}_{\text{BHL in step 10}} + \underbrace{s_j\, d_j(\#f_{j-1} + \#g_{j-1})}_{\text{VS in step 15}} + \underbrace{\#f_j\#g_j)}_{\text{step 19}}$$

$$\subseteq \mathcal{O}(d_js_j(\#f_{j-1} + \#g_{j-1} + d_1^2 + d_1d_j) + s_j\#a_j + \#f_j\#g_j)$$

$$= \mathcal{O}(d_js_j(T_{j-1} + d_1^2 + d_1d_j) + s_j\#a_j + \#f_j\#g_j).$$

□

For the whole process of CMSHL (for $2 \le j \le n$), we have the following:

Theorem 4.4.12. *Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ be monic in x_1. Let f and g be the monic irreducible factors of a. Let p be a large prime s.t. p does not divide any term of f and g.*

Let $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ *be a randomly chosen evaluation point from* $[1, \tilde{N} - 1]^{n-1}$. *Assume* $\boldsymbol{\alpha}$ *is Hilbertian. Let* $T_{fg} = \#f + \#g$. *Let* $d = \deg(a)$, $d_i = \deg(a, x_i)$ *for* $1 \leq i \leq n$, $d_{\max} = \max_{i=2}^{n}(d_i)$, *and* $s_{\max} = \max_{j=3}^{n}(s_j)$. *The failure probability of CMSHL is less than*

$$\frac{(n-2)(2d^2(s_{\max}^2 + T_{fg}) + ds_{\max}T_{fg})}{2(p-d+1)}. \tag{4.11}$$

Let $f_1 = f(x_1, \boldsymbol{\alpha})$ *and* $g_1 = g(x_1, \boldsymbol{\alpha})$ *be the image polynomials with* $\gcd(f_1, g_1) = 1$. *If CMSHL succeeds, the number of arithmetic operations in* $\mathbb{Z}_p$ *for lifting* f_1 *and* g_1 *to* f_n *and* g_n *in* $n-1$ *steps (via Algorithm 9) in the worst case is*

$$\mathcal{O}(\underbrace{d_1^2 d_2 + d_1 d_2^2}_{\text{first BHL}} + (n-2)\underbrace{(d_{\max}s_{\max}(T_{fg} + d_1^2 + d_1 d_{\max}) + s_{\max}\#a + \#f\#g)}_{\text{CMSHL } x_3, x_4, \cdots, x_n}). \tag{4.12}$$

Remark 1. In Theorem 4.4.12, $T_{fg} = \#f + \#g$ is the size of the output factors, which may be greater than $\#a$, the size of the input polynomial.

Chapter 5

Black box factorization algorithms

From the complexity analysis of CMSHL (Theorem 4.4.12), we observed that the dominating cost of CMSHL is often evaluating the input polynomial at many points. This motivated us to consider using a black box representation for the input polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$.

In this chapter, we present several algorithms for factoring polynomials represented by black boxes. We first discuss Kaltofen and Trager's algorithm [31] for factoring multivariate polynomials with coefficients in a field (Section 5.1), and then Rubinfeld and Zippel's algorithm [51] for factoring multivariate polynomials with integer coefficients (Section 5.2). We refer to both algorithms as Approach I (see Figure 1.4 in Section 1.7.1). For Approach I, the black boxes of the factors are constructed first, then the sparse representation of the factors is computed using *sparse polynomial interpolation* [2, 32, 62]. Both algorithms use a non-modular black box presentation of a.

The new algorithm CMBBSHL is presented in Section 5.3 (Approach II) for the monic and square-free case. Section 5.3 is based on our publication in 2022 [9]. Chapter 6 is based on our publication in ISSAC 2023 [10] which extends algorithm CMBBSHL to treat all cases of inputs, i.e. non-monic, non-square-free, and non-primitive cases. Algorithm CMBBSHL has been implemented in Maple with all major subroutines coded in C. Several benchmarks with a variety of problems are presented. We compared our timings with Maple and Magma's current best determinant and factorization algorithms and our algorithm is much faster when the determinant has large number of terms. We also show that the number of probes to the black box required in our algorithm is fewer than Rubinfeld and Zippel's algorithm [51] (the best known algorithm for Approach I), briefly in Section 5.3 and formally in the complexity analysis in Chapter 6.

5.1 Kaltofen and Trager's algorithm

Let $\mathsf{a} \in \mathbb{F}[x_1 \cdots, x_n]$, where $\mathbb{F}$ is a field of characteristic 0. The notation a is used here to distinguish polynomials over a field (only for this and the next sections). Let $\mathbf{BB} : \mathbb{F}^n \to \mathbb{F}$

Algorithm 10 Kaltofen and Trager's algorithm [31].

Input: (i) The black box $\mathbf{BB} : \mathbb{F}^n \to \mathbb{F}$ s.t. $\mathbf{BB}(\boldsymbol{\alpha}) = \mathsf{a}(\boldsymbol{\alpha})$; (ii) a given failure probability $\epsilon \ll 1$; (iii) $d = \deg(\mathsf{a})$, the total degree of a.

Output: $\mathbf{FF} : \mathbb{F}^n \to \mathbb{F}^r$ s.t. $\mathbf{FF}(\boldsymbol{\alpha}) = (f_1(\boldsymbol{\alpha}), \cdots, f_r(\boldsymbol{\alpha}))$.

Part *A***:** // This part computes $g_{1,i}(X_1)$ and their multiplicities $\bar{e}_i$, for all $1 \le i \le r$.

*A*1**:** Pick random elements $a_2, \cdots, a_n, b_1, \cdots, b_n, c_1, c_3, \cdots, c_n$ from a sufficiently large finite subset $R \subset \mathbb{F}$.

*A*2**:** By calling **BB** $\mathcal{O}(d^2)$ times and using Lagrange interpolation, interpolate $\mathsf{a}_2(X_1, X_2) := \mathsf{a}(X_1 + c_1X_2 + b_1, a_2X_1 + X_2 + b_2, a_3X_1 + c_3X_2 + b_3, \cdots, a_nX_1 + c_nX_2 + b_n)$.

*A*3**:** Factor a_2 over $\mathbb{F}[X_1, X_2]$ into irreducible factors

$$\mathsf{a}_2(X_1, X_2) = \prod_{i=1}^{\tilde{r}} g_{2,i}(X_1, X_2)^{\tilde{e}_i},\ \tilde{e}_i \in \mathbb{Z}^+.$$

$\Pr[\tilde{r} = r \text{ and } \tilde{e}_i = e_i] \ge 1 - \epsilon$ for all $1 \le i \le r$ (Theorem 1 of [31]).

*A*4**:** Assign $g_{1,i}(X_1) := g_{2,i}(X_1, 0)$ for all $1 \le i \le r$.

if $\gcd(g_{1,i}, g_{1,j}) \ne 1$ for any $1 \le i < j \le r$, **then** return FAIL.

if $\deg(g_{1,i}) \ne \deg(g_{2,i}, X_1)$ for any $1 \le i \le r$, **then** return FAIL.

Now, w.h.p.,

$$\mathsf{a}_1(X_1) := \mathsf{a}(X_1 + b_1, a_2X_1 + b_2, \cdots, a_nX_1 + b_n) = \prod_{i=1}^{r} g_{1,i}(X_1)^{e_i}. \tag{5.1}$$

Part *B***:** // The following program constructs the black box of the factors **FF**.

Input: $\boldsymbol{\alpha}$, r, $g_{1,i}$ $(1 \le i \le r)$, e_i $(1 \le i \le r)$, d, $a_2, \cdots, a_n$, $b_1, \cdots, b_n$, $c_1, c_3, \cdots, c_n$.

Output: $\mathbf{FF} : \mathbb{F}^n \to \mathbb{F}^r$ s.t. $\mathbf{FF}(\boldsymbol{\alpha}) = (f_1(\boldsymbol{\alpha}), \cdots, f_r(\boldsymbol{\alpha}))$.

procedure $(\boldsymbol{\alpha})$

*B*1**:** By calling **BB** $\mathcal{O}(d^2)$ times and using Lagrange interpolation, interpolate $\bar{\mathsf{a}}(X_1, Y) := \mathsf{a}(X_1 + b_1, Y(\alpha_2 - a_2(\alpha_1 - b_1) - b_2) + a_2X_1 + b_2, \cdots, Y(\alpha_n - a_n(\alpha_1 - b_1) - b_n) + a_nX_1 + b_n)$. Note that $\bar{\mathsf{a}}(\alpha_1 - b_1, 1) = \mathsf{a}(\underbrace{\alpha_1, \cdots, \alpha_n}_{\boldsymbol{\alpha}})$ and $\bar{\mathsf{a}}(X_1, 0) = \mathsf{a}_1(X_1)$.

*B*2**:** By Hensel lifting (5.1) obtain a factorization

$$\prod_{i=1}^{r} \bar{g}_i(X_1, Y)^{e_i} = \bar{\mathsf{a}}(X_1, Y) \pmod{Y^{d+1}},\ \deg(\bar{g}_i, Y) \le d.$$

Note that $\bar{g}_i(X_1, 0) = g_{1,i}(X_1)$ for all $1 \le i \le r$. Test if $\bar{g}_i$ divides $\bar{\mathsf{a}}$ for all $1 \le i \le r$. If any test fails, return 'program incorrect', since the factor degree patterns of a and a_2 must disagree.

*B*3**:** return $\bar{g}_i(\alpha_1 - b_1, 1) = f_i(\boldsymbol{\alpha})$ for $1 \le i \le r$.

end procedure

be the black box representation of a. Suppose

$$\mathsf{a}(x_1, \cdots, x_n) = \prod_{\rho=1}^{r} f_\rho(x_1, \cdots, x_n)^{e_\rho},\ e_\rho \in \mathbb{Z}^+,$$

is the factorization of a into irreducibles. Let $\mathbf{FF} : \mathbb{F}^n \to \mathbb{F}^r$ be a black box that represents the irreducible factors f_ρ $(1 \le \rho \le r)$ s.t. $\mathbf{FF}(\boldsymbol{\alpha}) = (f_1(\boldsymbol{\alpha}), \cdots, f_r(\boldsymbol{\alpha}))$.

Kaltofen and Trager's algorithm uses a bivariate transformation to compute the evaluations of the factors. Their algorithm for constructing the black boxes of the factors is summarized in Algorithm 10. It consists of two parts: Part A and Part B. Part A computes a list of factors of a bivariate image of a, and their multiplicities $\bar{e}_\rho \in \mathbb{Z}^+$ s.t. with a probability greater or equal than $1 - \epsilon$, $\bar{e}_\rho = e_\rho$ for all $1 \le \rho \le r$. Using the factors computed in Part A for determining the order of the factors, Part B outputs the black box $\mathbf{FF}$ w.h.p. The failure probability applies to the construction of the black box, not to the execution of the black box $\mathbf{FF}$. If the black box $\mathbf{FF}$ is constructed correctly, it always produces the true values of the factors, up to a fixed associate for each factor.

The input and output of Algorithm 10 are:

- Input:
  (i) The black box $\mathbf{BB} : \mathbb{F}^n \to \mathbb{F}$ s.t. $\mathbf{BB}(\boldsymbol{\alpha}) = \mathsf{a}(\boldsymbol{\alpha})$;
  (ii) a failure probability $\epsilon \ll 1$;
  (iii) $d = \deg(\mathsf{a})$, the total degree of a.

- Output:
  The black box of the factors $\mathbf{FF} : \mathbb{F}^n \to \mathbb{F}^r$ s.t. $\mathbf{FF}(\boldsymbol{\alpha}) = (f_1(\boldsymbol{\alpha}), \cdots, f_r(\boldsymbol{\alpha}))$.

The multivariate polynomial a is first reduced to a bivariate polynomial $\mathsf{a}_2(X_1, X_2)$ in step A2. Then, the second variable X_2 is set to 0 to further reduce it to a univariate polynomial $\mathsf{a}_1(X_1)$. Reduction to a bivariate polynomial first is necessary for a general field $\mathbb{F}$ since their algorithm uses the effective Hilbert irreducibility theorem [30] to ensure the factor degree patterns of a and a_2 agree w.h.p., i.e. $e_\rho = \tilde{e}_\rho$ w.h.p., for all $1 \le \rho \le r$.

5.1.1 Number of probes to the black box

Theorem 5.1.1. *(Theorem 1 in [31]) Let $\mathbf{BB} : \mathbb{F}^n \to \mathbb{F}$ be the black box representation of $\mathsf{a} \in \mathbb{F}[x_1, \cdots, x_n]$, where $\mathbb{F}$ is a field of characteristic 0. Let $d = \deg(\mathsf{a})$. Kaltofen and Trager's method (Algorithm 10) can construct the black box of the factors of a in polynomially many arithmetic operations in $\mathbb{F}$ as a function of n and d, and an additional bivariate polynomial factorization in $\mathbb{F}[X_1, X_2]$ (or a univariate polynomial factorization in $\mathbb{F}[X_1]$ plus a bivariate Hensel lifting). It requires $\mathcal{O}(d^2)$ probes to $\mathbf{BB}$. If the cardinality of R in step A1 of Algorithm 10 satisfies*

$$card(R) \ge 6d2^d/\epsilon,$$

then the algorithm succeeds with a probability no less than $1-\epsilon$ and the output program, the black box of the factors $\mathbf{FF} : \mathbb{F}^n \to \mathbb{F}^r$, always correctly evaluate all irreducible factors of a, i.e. $\mathbf{FF}(\boldsymbol{\alpha}) = (f_1(\boldsymbol{\alpha}), \cdots, f_r(\boldsymbol{\alpha}))$ (up to the same units).

The output black box **FF** *can be executed in polynomially many arithmetic operations in* $\mathbb{F}$*. It performs one bivariate polynomial factorization (or one univariate polynomial factorization in* $\mathbb{F}[X_1]$ *plus one bivariate Hensel lifting). It calls the black box* **BB** $\mathcal{O}(d^2)$ *times.*

Theorem 5.1.2. *Let* **FF** $: \mathbb{F}^n \rightarrow \mathbb{F}^r$ *denote the black box of the irreducible factors of* $\mathsf{a} \in \mathbb{F}[x_1, \cdots, x_n]$ *constructed by Kaltofen and Trager's algorithm (Algorithm 10). Let* $\delta_j = \max_{\rho=1}^{r} \deg(f_\rho, x_j)$*,* $\delta_{\max} = \max_{j=1}^{n} \delta_j$ *and* $\#f_{\max} = \max_{\rho=1}^{r} \#f_\rho$*. Using Zippel's sparse interpolation [62], it takes* $\mathcal{O}(n\delta_{\max} d^2 \#f_{\max})$ *probes to* **BB** *plus* $\mathcal{O}(n\delta_{\max}\#f_{\max})$ *bivariate polynomial factorizations to recover the sparse representation of the irreducible factors* f_ρ *for all* $1 \leq \rho \leq r$*.*

Remark 2. If instead one uses Ben-Or and Tiwari's sparse interpolation algorithm [2], it only takes $\mathcal{O}(d^2 \#f_{\max})$ probes to the black box **BB** plus $\mathcal{O}(\#f_{\max})$ bivariate polynomial factorizations to recover the sparse representation of the factors. However, for Ben-Or and Tiwari's algorithm, a much bigger prime is required than Zippel's sparse interpolation thus large integer arithmetic operations are used.

Remark 3. If $\mathbb{F} = \mathbb{Q}$, an expression swell occurs also in the Hensel lifting wherein one must solve Diophantine equations in $\mathbb{Q}[x]$ (at step $B2$ of Algorithm 10) during the construction of the black box **FF**. One of the reasons that our algorithm is much faster in practice is that we work modulo a prime.

5.2 Rubinfeld and Zippel's algorithm

Rubinfeld and Zippel's algorithm [51] does not apply any bivariate transformation as in Kaltofen and Trager's method [31]. Since it is designed specifically for factoring polynomials with coefficients in $\mathbb{Q}$ (and hence for $\mathbb{Z}$), it uses a simple evaluation point for each variable thus the *classical* Hilbert irreducibility theorem [34] (see also Section 3.2) is applied. Their algorithm does not use the *effective* Hilbert irreducibility theorem [30] as in Kaltofen and Trager's algorithm.

Let **BB** $: \mathbb{Q}^{n+1} \rightarrow \mathbb{Q}$ be the black box representation of $\mathsf{a} \in \mathbb{Q}[X, Y_1, \cdots, Y_n]$. For Rubinfeld and Zippel's algorithm [51], a is assumed to be square-free and monic in X. For non-monic polynomials, a simple modification can be made (described in Appendix C of [51]) so that the algorithm for the monic case can still be used. Suppose the irreducible factorization of a is

$$\mathsf{a} = \prod_{\rho=1}^{r} f_\rho(X, Y_1, \cdots, Y_n)$$

where r is the number of irreducible factors of a. Let $\delta_0 = \max_{\rho=1}^{r} \deg(f_\rho, X)$. Expanding the factors, we get

$$\begin{aligned} f_1 &= p_{1,\delta_0}(Y_1, \cdots, Y_n)X^{\delta_0} + p_{1,\delta_0-1}(Y_1, \cdots, Y_n)X^{\delta_0-1} + \cdots + p_{1,0}(Y_1, \cdots, Y_n), \\ f_2 &= p_{2,\delta_0}(Y_1, \cdots, Y_n)X^{\delta_0} + p_{2,\delta_0-1}(Y_1, \cdots, Y_n)X^{\delta_0-1} + \cdots + p_{2,0}(Y_1, \cdots, Y_n), \\ &\vdots \\ f_r &= p_{r,\delta_0}(Y_1, \cdots, Y_n)X^{\delta_0} + p_{r,\delta_0-1}(Y_1, \cdots, Y_n)X^{\delta_0-1} + \cdots + p_{r,0}(Y_1, \cdots, Y_n). \end{aligned} \tag{5.2}$$

Since f_ρ's are monic in X, we have for $1 \leq \rho \leq r$,

$$p_{\rho,\delta_0} = \begin{cases} 1 \text{ if } \deg(f_\rho, X) = \delta_0, \\ 0 \text{ if } \deg(f_\rho, X) < \delta_0. \end{cases}$$

The first step of Rubinfeld and Zippel's algorithm is to build $\delta_0 + 1$ black boxes $\mathbf{M}_i$ $(0 \leq i \leq \delta_0)$ for the coefficients of f_ρ $(1 \leq \rho \leq r)$ in X^i, i.e. $\mathbf{M}_i : \mathbb{Q}^n \to \mathbb{Q}^r$ s.t. $\mathbf{M}_i(\boldsymbol{y}) = \{p_{1,i}(\boldsymbol{y}), \cdots, p_{r,i}(\boldsymbol{y})\}$ for all $0 \leq i \leq \delta_0$, where $\boldsymbol{y} = (y_1, \cdots, y_n) \in \mathbb{Q}^n$ and $\boldsymbol{y}$ is chosen randomly from $[0, \tilde{N})^n$, a sufficiently large set to make $\boldsymbol{y}$ Hilbertian (defined in Section 3.2) w.h.p. The assumption that $\boldsymbol{y}$ is Hilbertian ensures that the irreducible factors of a evaluated at $\boldsymbol{y}$ remain irreducible. Note that each $\mathbf{M}_i$ is an unordered black box meaning that $\{p_{\rho,1}(\boldsymbol{y}), \cdots, p_{\rho,r}(\boldsymbol{y})\}$ is an unordered set. We can construct the black boxes $\mathbf{M}_i$ by factoring $\mathsf{a}(X, \boldsymbol{y})$ into irreducibles. If a bound for $d_1 = \deg(\mathsf{a}, X)$ is known, $\mathsf{a}(X, \boldsymbol{y})$ can be interpolated by calling the black box $\mathbf{BB}$ $d_1 + 1$ times and using a Lagrange interpolation. This process is called a *univariate dense interpolation*.

Secondly, the unordered black boxes $\mathbf{M}_i$ are transformed into ordered black boxes $\mathbf{M}'_i : \mathbb{Q}^n \to \mathbb{Q}^r$ s.t. $\mathbf{M}'_i(\boldsymbol{y}) = (p_{1,i}(\boldsymbol{y}), \cdots, p_{r,i}(\boldsymbol{y}))$ for all $0 \leq i \leq \delta_0$. The transformation involves a technique of interpolating an unordered multi-valued black box of univariate polynomials, which we shall discuss in Section 5.2.2.

Lastly, the ordered black boxes $\mathbf{M}'_i$ $(0 \leq i \leq \delta_0)$ can be interpolated by a standard sparse multivariate interpolation algorithm [2, 32, 62].

5.2.1 Black boxes of multivariate polynomials

Given an unordered black box $\mathbf{M} : \mathbb{Q}^n \to \mathbb{Q}^r$ representing the multivariate polynomials $p_1, \cdots, p_r \in \mathbb{Q}[Y_1, \cdots, Y_n]$, we aim to convert it to an ordered black box $\mathbf{M}' : \mathbb{Q}^n \to \mathbb{Q}^r$ s.t. $\mathbf{M}'(\boldsymbol{y}) = (p_1(\boldsymbol{y}), \cdots, p_r(\boldsymbol{y}))$.

Let $\boldsymbol{z} = (z_1, \cdots, z_n) \in \mathbb{Q}^n$ be a reference point. We say $\boldsymbol{z}$ is a *reference point* if $p_i(\boldsymbol{z}) \neq p_j(\boldsymbol{z})$ for all $i \neq j$. Given the reference point $\boldsymbol{z}$, we compute one ordered sequence

$$\mathbf{M}(\boldsymbol{z}) = (p_1(\boldsymbol{z}), \cdots, p_r(\boldsymbol{z})).$$

Define an unordered black box for univariate polynomials $\mathbf{U}_{\boldsymbol{y},\boldsymbol{z}}$ as $\mathbf{U}_{\boldsymbol{y},\boldsymbol{z}} : \mathbb{Q} \to \mathbb{Q}^r$ s.t. $\mathbf{U}_{\boldsymbol{y},\boldsymbol{z}}(\theta) = \mathbf{M}(\boldsymbol{z} + \theta(\boldsymbol{y} - \boldsymbol{z})) = \{p_1(\boldsymbol{z} + \theta(\boldsymbol{y} - \boldsymbol{z})), \cdots, p_r(\boldsymbol{z} + \theta(\boldsymbol{y} - \boldsymbol{z}))\}$.

For a fixed $\boldsymbol{y}$, applying the technique described in Section 5.2.2, we can get explicitly the set of univariate polynomials

$$S_{\boldsymbol{y}} = \{p_1(\boldsymbol{z} + \Theta(\boldsymbol{y} - \boldsymbol{z})), \cdots, p_r(\boldsymbol{z} + \Theta(\boldsymbol{y} - \boldsymbol{z}))\}.$$

By substituting $\Theta = 0$ into $S_{\boldsymbol{y}}$ and comparing it with the reference value $\mathbf{M}(\boldsymbol{z})$, we determine the order of $p_1, \cdots p_r$. By substituting $\Theta = 1$, we can now determine $p_1(\boldsymbol{y}), \cdots, p_r(\boldsymbol{y})$.

5.2.2 Black boxes of univariate polynomials

Let $p_1(\Theta), \cdots, p_r(\Theta)$ be univariate polynomials with degree no more than D. Given an unordered black box $\mathbf{U}$ s.t. $\mathbf{U}(\theta) = \{p_1(\theta), \cdots, p_r(\theta)\}$, the goal is to reconstruct the polynomials $p_1(\Theta), \cdots, p_r(\Theta)$.

Define the symmetric functions in the $p_1(\theta), \cdots, p_r(\theta)$ to be

$$\begin{aligned}
\sigma_1(\theta) &= p_1(\theta) + p_2(\theta) + \cdots + p_r(\theta), \\
\sigma_2(\theta) &= p_1(\theta)p_2(\theta) + p_1(\theta)p_3(\theta) + \cdots + p_{r-1}(\theta)p_r(\theta), \\
&\vdots \\
\sigma_r(\theta) &= p_1(\theta)p_2(\theta)\cdots p_r(\theta).
\end{aligned}$$

The symmetric functions can be computed given the values of $\{p_1(\theta), \cdots, p_r(\theta)\}$ without knowing which one is which. Now we can construct r black boxes $\mathbf{\Sigma}_\rho$, each for the symmetric function $\sigma_\rho(\Theta)$, i.e. $\mathbf{\Sigma}_\rho : \mathbb{Q} \to \mathbb{Q}$ s.t. $\mathbf{\Sigma}_\rho(\theta) = \sigma_\rho(\theta)$ for all $1 \le \rho \le r$. Each $\sigma_\rho(\Theta)$ has degree no more than ρD and can be interpolated using Lagrange interpolation in $\mathcal{O}((\rho D)^2)$ arithmetic operations in $\mathbb{Q}$ and $\mathcal{O}(\rho D)$ calls to $\mathbf{\Sigma}_\rho$.

After determining the $\sigma_\rho(\Theta)$'s, the following bivariate polynomial

$$Q(Z, \Theta) = Z^r - \sigma_1(\Theta)Z^{r-1} + \sigma_2(\Theta)Z^{r-2} + \cdots + (-1)^r\sigma_r(\Theta)$$

can be constructed. It factors into linear factors as

$$Q(Z, \Theta) = (Z - p_1(\Theta))(Z - p_2(\Theta))\cdots(Z - p_r(\Theta)).$$

Hence $p_1(\Theta), \cdots, p_r(\Theta)$ are determined.

5.2.3 The number of probes to the black box

Theorem 5.2.1. *Let* $\mathbf{BB} : \mathbb{Q}^{n+1} \to \mathbb{Q}$ *be the black box representing* $\mathsf{a} \in \mathbb{Q}[X, Y_1, \cdots, Y_n]$. *Suppose* a *can be factored into irreducible factors as represented in (5.2). Let* $T_{\max}$ *be the*

maximum number of terms in all $p_{\rho,i}$ for all $1 \leq \rho \leq r$ and for all $0 \leq i \leq \delta_0$. Let $d_1 = \deg(\mathsf{a}, X)$. Let $\delta_j = \max_{\rho=1}^{r} \deg(f_\rho, Y_j)$ and $\delta_{\max} = \max_{j=1}^{n} \delta_j$. If using Zippel's sparse interpolation [62] to recover the sparse representation of the factors f_ρ, the required number of probes to **BB** *for Rubinfeld and Zippel's algorithm [51] is $\mathcal{O}(rn^2\delta_{\max}^2 d_1 T_{\max})$. Also, it does $\mathcal{O}(rn^2\delta_{\max}^2 T_{\max})$ univariate polynomial factorizations in $\mathbb{Q}[X]$.*

I present a proof below since my result for the number of probes to the black box in Theorem 5.2.1 is fewer than in [51].

Proof. To produce one ordered evaluation of $\mathbf{M}'_i$ $(0 \leq i \leq \delta_0)$ from the unordered black box $\mathbf{M}_i$, Rubinfeld and Zippel's algorithm needs to call $\mathbf{M}_i$ and hence $\mathbf{U}_{\boldsymbol{y},\boldsymbol{z}}$ exactly $rn\delta_{\max} + 1$ times. This is because the degree (in Θ) of each $p_{\rho,i}(\boldsymbol{z} + \Theta(\boldsymbol{y} - \boldsymbol{z}))$ for $1 \leq \rho \leq r$ is no more than $n\delta_{\max}$. Thus, it requires $rn\delta_{\max}+1$ probes to $\mathbf{U}_{\boldsymbol{y},\boldsymbol{z}}$ to determine each $p_{\rho,i}(\boldsymbol{z}+\Theta(\boldsymbol{y}-\boldsymbol{z}))$, and hence $(p_{1,i}(\boldsymbol{y}), \cdots, p_{r,i}(\boldsymbol{y}))$ as an ordered output of $\mathbf{M}'_i$ $(0 \leq i \leq \delta_0)$.

By probing the black box **BB** $d_1 + 1$ times and doing a Lagrange interpolation in the variable X, we can get a univariate image $\mathsf{a}(X, \boldsymbol{y})$. Factoring it produces one evaluation for all the $\mathbf{M}_i$'s. Thus, to get one evaluation of $\mathbf{M}'_i$ for all $0 \leq i \leq \delta_0$, we need $\Theta(rn\delta_{\max}d_1)$ probes to **BB** and $\Theta(rn\delta_{\max})$ univariate polynomial factorizations.

If using Zippel's sparse interpolation [62], it requires $\mathcal{O}(n\delta_{\max}T_{\max})$ evaluation points to interpolate all $p_{\rho,i}$'s and hence all the factors. Thus, the total number of probes to **BB** is $\mathcal{O}(rn^2\delta_{\max}^2 d_1 T_{\max})$. The total number of univariate factorizations is $\mathcal{O}(rn^2\delta_{\max}^2 T_{\max})$. □

Remark 4. If using Ben-Or and Tiwari's sparse interpolation algorithm [2], it only takes $\mathcal{O}(rn\delta_{\max}d_1T_{\max})$ probes to the black box **BB** plus $\mathcal{O}(rn\delta_{\max}T_{\max})$ univariate polynomial factorizations to recover the sparse representation of the factors. However, a very big prime is needed for Ben-Or and Tiwari's algorithm as the black box **BB** is non-modular thus large integer arithmetic operations are used.

5.3 My new algorithm CMBBSHL: monic and square-free case

Both algorithms described in Section 5.1 and Section 5.2 are referred to as Approach I in Figure 1.4 and they both use a non-modular black box representation of $\mathsf{a} \in \mathbb{Q}[x_1, \cdots, x_n]$. Unlike Rubinfeld and Zippel's algorithm [51] or Kaltofen and Trager's algorithm [31], my new algorithm CMBBSHL uses a modular black box representation of $a \in \mathbb{Z}[x_1, \cdots, x_n]$. All arithmetic operations are performed in $\mathbb{Z}_p$, so my algorithm avoids large integer arithmetic operations. Let $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ denote the modular black box representation of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$. For simplicity, we first consider the case of a being monic in x_1 (the chosen main variable) and square-free. The more difficult non-monic, non-square-free, and non-primitive cases are discussed in Chapter 6. Since a is monic in x_1, a has no content and we do not need to compute it. Also, since a is square-free, we can simply

extend Algorithm 9 (CMSHL) to work with multi-factors and adapt it to the black box case.

The following steps are performed prior to Hensel lifting:

1. Choose a large prime p, e.g. $p = 2^{62} - 57$ and a positive integer $\tilde{N} < p$.

2. Choose an evaluation point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ randomly from a sufficiently large set $[1, \tilde{N} - 1]^{n-1}$. I use $\tilde{N} = 100003$ for computing the factors of $\det(T_n)$, where T_n is an $n \times n$ symmetric Toeplitz matrix.

3. Compute $d_j = \deg(a, x_j)$ for all $1 \leq j \leq n$ w.h.p. (e.g. by Algorithm 4).

4. $a(x_1, \boldsymbol{\alpha}) \bmod p$ is computed w.h.p. by a univariate dense interpolation. Then $a(x_1, \boldsymbol{\alpha}) \in \mathbb{Z}[x_1]$ is computed using Chinese remaindering with different primes to recover the coefficients of $a(x_1, \boldsymbol{\alpha})$ in $\mathbb{Z}$.

5. $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}$. Let $a = f_1 f_2 \cdots f_r$ be the irreducible factorization of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ over $\mathbb{Z}$. Then, w.h.p. $f_\rho(x_1, \boldsymbol{\alpha})$ are irreducible in $\mathbb{Z}[x_1]$ for all $1 \leq \rho \leq r$, if $\tilde{N}$ is sufficiently large.

Define $f_{\rho,1} := f_\rho(x_1, \boldsymbol{\alpha}) \bmod p$ $(1 \leq \rho \leq r)$ and $a_j := a(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ for $1 \leq j \leq n$. Let $f_{\rho,j} := f_\rho(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ $(2 \leq \rho \leq r)$. The input and output of sparse Hensel lifting algorithm CMBBSHL for the monic and square-free case is:

- Input: A prime p, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, the modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$ (a is monic in x_1 and square-free), $d_i = \deg(a, x_i)$ $(1 \leq i \leq n)$ (pre-computed), $f_{\rho,1} \in \mathbb{Z}_p[x_1]$ $(1 \leq \rho \leq r)$ s.t.
  (i) $\gcd(f_{k,1}, f_{l,1}) = 1$ for $k \neq l$ in $\mathbb{Z}_p[x_1]$, for Hensel lifting to work,
  (ii) $a_1 = a(x_1, \boldsymbol{\alpha}) \bmod p = \prod_{\rho=1}^{r} f_{\rho,1} \in \mathbb{Z}_p[x_1]$,
  (iii) $f_{\rho,1}$ is monic in x_1 for all $1 \leq \rho \leq r$.

- Output: $f_{\rho,n} \in \mathbb{Z}_p[x_1, \cdots, x_n]$ $(1 \leq \rho \leq r)$ s.t.
  (i) $a_n = \prod_{\rho=1}^{r} f_{\rho,n} \in \mathbb{Z}_p[x_1, \cdots, x_n]$,
  (ii) $f_{\rho,n}(x_1, \boldsymbol{\alpha}) = f_{\rho,1}$ for $1 \leq \rho \leq r$;
  Or FAIL (at step 4, 11 or 23 of Algorithm 11).

Other than returning FAIL with a low probability, algorithm CMBBSHL could also return an incorrect answer with a low probability. The analysis for failure probabilities is presented in Chapter 6. Algorithm CMBBSHL lifts $f_{\rho,1}(x_1)$ to $f_{\rho,2}(x_1, x_2)$ then $f_{\rho,2}(x_1, x_2)$ to $f_{\rho,3}(x_1, x_2, x_3)$ etc. until we obtain $f_{\rho,n}(x_1, \cdots, x_n)$ for $1 \leq \rho \leq r$. At each step $a_j = \prod_{\rho=1}^{r} f_{\rho,j}$, and at the end $a_n = \prod_{\rho=1}^{r} f_{\rho,n}$. If p is not large enough to recover the integer coefficients in f_ρ, we need to use larger and larger primes until the the algorithm succeeds.

The j^{th} Hensel lifting step is shown in Algorithm 11. The number of arithmetic operations in $\mathbb{Z}_p$ for these steps is shown in blue. There are four major sub-steps, namely

1. the probes to the black box to interpolate the bivariate images $A_k = a(x_1, Y_k, x_j)$ in step 8,

2. evaluations of the factors $f_{\rho,j-1}$ for $1 \leq \rho \leq r$ in step 10,

3. the bivariate Hensel lifts [4, 49] in step 12,

4. solving the Vandermonde systems [62] in step 18.

All four steps are parallelizable.

The major difference between Algorithm 11 and Algorithm 9 is at step 8 of Algorithm 11 (highlighted in blue). Instead of evaluating the polynomial a (step 8 of Algorithm 9), the bivariate images of a, $A_k(x_1, x_j)$, are interpolated from the black box **B** using **bivariate dense interpolations**, i.e. using Lagrange (or Newton) interpolations for a bivariate polynomial (see Chapter 7 for a more detailed description).

The test to check if $a_j = \prod_{\rho=1}^{r} f_{\rho,j}$ at the end of the j^{th} Hensel lifting step becomes probabilistic (step 23). This avoids explicit multiplications of the factors to obtain the product, $\prod_{\rho=1}^{r} f_{\rho,j}$.

5.3.1 Number of probes to the black box

In order to count the number of probes to the black box for algorithm CMBBSHL and compare with the algorithms for Approach I, we define an important quantity, $s_{\max}$, in Definition 5.3.1.

Definition 5.3.1. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$. Let $(f_\rho, 1 \leq \rho \leq r)$ be the irreducible factors of a over $\mathbb{Z}$. Let $f_\rho = \sum_{i=0}^{df_\rho} \sigma_{\rho,i}(x_2, \cdots, x_n)x_1^i$, where $df_\rho = \deg(f_\rho, x_1)$. Define

$$s_{\max} = \max_{\rho=1}^{r} \max_{i=0}^{df_\rho} \#\sigma_{\rho,i}. \tag{5.3}$$

Example 13. Let $n = 3$, $\rho = 2$,

$$f_1 = (7x_2^2 + 31x_3)x_1^3 + (5x_2^2x_3 - 29x_3)x_1^2 + (97x_3^3 + 1)x_1 + (2x_2x_3^2 + 58), \text{ and}$$
$$f_2 = (-36x_2x_3^2 + 5x_3)x_1 + (47x_2x_3^2 + 90x_2).$$

In this case, $\max_i \#\sigma_{1,i} = 2 = 8/(3+1) = \#f_1/(df_1+1)$, and $\max_i \#\sigma_{2,i} = 2 = 4/(1+1) = \#f_2/(df_2+1)$. Thus, $s_{\max} = 2$.

Example 14. $f_1 = (3x_2x_3)x_1 + 49x_2$, and $f_2 = x_1^4 + (-63x_2^3 - 5x_2^2x_3 + 3x_3^3 + 78x_2^2 + 5x_3^2 - 12x_3 + 65)$. Now, $s_{\max} = 7 = \max_i \#\sigma_{2,i} = \#f_2 - 1$.

Proposition 5.3.2. *Assume f_ρ has at least two terms in x_1. Then,*

$$\max_{\rho=1}^{r} \frac{\#f_\rho}{df_\rho + 1} \leq s_{\max} \leq \max_{\rho=1}^{r} \#f_\rho - 1. \tag{5.4}$$

Algorithm 11 CMBBSHL for black box: Hensel lifting x_j (square-free and monic).

Input: A prime p, $\alpha_j \in \mathbb{Z}$, the modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t.
$\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$, $d_i = \deg(a, x_i)$ $(1 \le i \le n)$ (pre-computed),
$f_{\rho,j-1} \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$ $(1 \le \rho \le r)$ s.t. $a_j(x_j = \alpha_j) = \prod_{\rho=1}^{r} f_{\rho,j-1}$ with $j > 2$.
Output: $f_{\rho,j} \in \mathbb{Z}_p[x_1, \cdots, x_j]$ $(1 \le \rho \le r)$ s.t.
(i) $a_j = \prod_{\rho=1}^{r} f_{\rho,j}$, (ii) $f_{\rho,j}(x_j = \alpha_j) = f_{\rho,j-1}$ for $1 \le \rho \le r$;
Otherwise, **return** FAIL.

1: Let $f_{\rho,j-1} = x_1^{df_\rho} + \sum_{i=0}^{df_\rho - 1} \sigma_{\rho,i}(x_2, ..., x_{j-1}) x_1^i$ where $\sigma_{\rho,i} = \sum_{k=1}^{s_{\rho,i}} c_{\rho,ik} M_{\rho,ik}$, $M_{\rho,ik}$ are the monomials in $\sigma_{\rho,i}$ for $1 \le \rho \le r$. $df_\rho = \deg(f_{\rho,j-1}, x_1)$.
2: Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
3: Evaluate: $\{\mathcal{S}_\rho = \{S_{\rho,i} = \{m_{\rho,ik} = M_{\rho,ik}(\boldsymbol{\beta}), 1 \le k \le s_{\rho,i}\}, 0 \le i \le df_\rho - 1\}, 1 \le \rho \le r\}$.
4: **if** any $|S_{\rho,i}| \ne s_{\rho,i}$ **then return** FAIL **end if**
5: Let s be the maximum of $s_{\rho,i}$.
6: **for** k from 1 to s **do**
7: Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
8: $A_k \leftarrow a_j(x_1, Y_k, x_j) \in \mathbb{Z}_p[x_1, x_j]$. // via probes to $\mathbf{B}$ and bivariate dense interpolation ..... $\mathcal{O}(sd_1d_j\mathrm{C}(\text{probe } \mathbf{B}) + s(d_1^2 d_j + d_1 d_j^2))$
9: **if** $\deg(A_k, x_j) \ne d_j$ **then return** FAIL **end if**
10: $F_{\rho,k} \leftarrow f_{\rho,j-1}(x_1, Y_k) \in \mathbb{Z}_p[x_1]$ for $1 \le \rho \le r$. ..... $\mathcal{O}\left(s\left(\sum_{\rho=1}^{r} \#f_{\rho,j-1}\right)\right)$
11: **if** $\gcd(F_{\rho,k}, F_{\phi,k}) \ne 1$ for any $\rho \ne \phi$ $(1 \le \rho, \phi \le r)$ **then return** FAIL **end if**
12: $f_{\rho,k} \leftarrow \mathrm{BivariateHenselLift}(A_k(x_1, x_j), F_{\rho,k}(x_1), \alpha_j, p)$. ..... $\mathcal{O}(s(d_1 d_j^2 + d_1^2 d_j))$
13: **end for**
14: Let $f_{\rho,k} = x_1^{df_\rho} + \sum_{l=1}^{t_\rho} \alpha_{\rho,kl} \tilde{M}_{\rho,l}(x_1, x_j)$ for $1 \le k \le s$ where $t_\rho \le d_1 d_j$ for $1 \le \rho \le r$.
15: **for** ρ from 1 to r **do**
16: **for** l from 1 to t_ρ **do**
17: $i \leftarrow \deg(\tilde{M}_{\rho,l}, x_1)$.
18: Solve the linear system for $c_{\rho,lk}$: $\left\{\sum_{k=1}^{s_{\rho,i}} m_{\rho,ik}^t c_{\rho,lk} = \alpha_{\rho,tl} \text{ for } 1 \le t \le s_{\rho,i}\right\}$.
19: **end for** ..... $\mathcal{O}\left(sd_j\left(\sum_{\rho=1}^{r} \#f_{\rho,j-1}\right)\right)$
20: Construct $f_{\rho,j} \leftarrow x_1^{df_\rho} + \sum_{l=1}^{t_\rho} \left(\sum_{k=1}^{s_{\rho,i}} c_{\rho,lk} M_{\rho,ik}(x_2, ..., x_{j-1})\right) \tilde{M}_{\rho,l}(x_1, x_j)$.
21: **end for**
22: Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_j) \in \mathbb{Z}_p^{j-1}$ at random.
23: **if** $\mathbf{B}(\boldsymbol{\beta}, \alpha_{j+1}, \cdots, \alpha_n) = \prod_{\rho=1}^{r} f_{\rho,j}(\boldsymbol{\beta})$ **then return** $f_{\rho,j}$ $(1 \le \rho \le r)$ **else return** FAIL
24: **end if**

Proof. Since

$$\frac{\#f_\rho}{df_\rho + 1} \le \max_{i=0}^{df_\rho} \#\sigma_{\rho,i} \le \#f_\rho - 1,$$

for all $1 \le \rho \le r$, the inequality in (5.4) follows from Definition 5.3.1. □

Since step 8 of Algorithm 11 is the only step to probe the black box, the number of probes to the black box $\mathbf{B}$ needed for algorithm CMBBSHL is given by Theorem 5.3.3.

Theorem 5.3.3. *Let* $\mathbf{B}$ *be the modular black box representation of* $a \in \mathbb{Z}[x_1, \cdots, x_n]$ *where* a *is square-free and monic in* x_1. *Let* $d_j = \deg(a, x_j)$ *for* $1 \le j \le n$ *and* $d_{\max} = \max_{j=2}^{n} d_j$.

The total number of probes to the black box $\mathbf{B}$ *for CMBBSHL is*

$$\sum_{j=2}^{n} s_j(d_1+1)(d_j+1) \in \mathcal{O}(nd_1d_{\max}s_{\max}). \tag{5.5}$$

Proof. Let s_j be he number s defined in step 5 of Algorithm 11. For step 8, using bivariate dense interpolation (for details, see Chapter 7) to interpolate the bivariate image $A_k(x_1, x_j) = a_j(x_1, Y_k, x_j)$ we need $(d_1+1)(d_j+1)$ points. Thus, the total number of probes to $\mathbf{B}$ for step j is $\mathcal{O}(d_1d_js_j)$. Hence, the total number of probes to the black box $\mathbf{B}$ for algorithm 11 is $\mathcal{O}((n-1)d_1d_{\max}s_{\max}) = \mathcal{O}(nd_1d_{\max}s_{\max})$. □

For Rubinfeld and Zippel's method [51], the total number of probes to the black box is $\mathcal{O}(rn^2\delta_{\max}^2 d_1 T_{\max})$, where $\delta_{\max}$ is the maximum degree in each variable in all the factors, $T_{\max}$ is the maximum number of terms in all the coefficients of x_1 in all the factors. We have $T_{\max} \geq s_{\max}$ and $r\delta_{\max} \geq d_{\max}$, so Rubinfeld and Zippel's algorithm probes the black box at least $\mathcal{O}(n\delta_{\max})$ times more than our algorithm. Second, Rubinfeld and Zippel's algorithm evaluates over $\mathbb{Z}$ and my algorithm performs arithmetic operations in $\mathbb{Z}_p$. Their algorithm has to do arithmetic operations with large integers, and there is a large loss of efficiency because of this. Third, their algorithm does $\mathcal{O}(rn^2\delta_{\max}^2 T_{\max})$ univariate factorizations, which is a large number and will be the bottleneck. My new algorithm has only one univariate factorization to do at the beginning.

5.3.2 Benchmarks

We made a hybrid Maple + C implementation for Algorithm 11. The main program is in Maple and the major subroutines are coded in C. To call C code from Maple we use Maple's foreign function interface (see Section 7.1 for details). This allows us to pass arrays of 32-bit or 64-bit integers between Maple and C. Our Maple code and C code can be downloaded from `http://www.cecm.sfu.ca/~mmonagan/code/CMSHL/`. Instructions for compiling the C code are given there.

In order to get better performance each of the four main sub-steps has been coded in C. Step 8 interpolates the bivariate image $A_k(x_1, x_j) = a_j(x_1, Y_k, x_j)$ by probing the black box at $(d_1+1)(d_j+1)$ points and a bivariate dense interpolation. Both evaluations of the black box and bivariate dense interpolations are coded in C. I coded the bivariate dense interpolation in C. For the problem of computing the determinant of a matrix A, the black box $\mathbf{B}$ consists of two parts: *BB eval* and *BB det*. *BB eval* evaluates the polynomial entries of the matrix A (coded in C by Prof. Monagan). *BB det* computes the determinant of the evaluated entries in $\mathbb{Z}_p$ by Gaussian Elimination (coded in C by Prof. Monagan). I assembled the C codes *BB eval* and *BB det* to make a black box that is called from Maple.

n	10	11	12	13	14	15	16
CMBBSHL	5.790	13.430	50.855	154.441	722.310	1967.725	17,212.991
# probes	109,139	267,465	894,358	2,180,399	6,981,462	17,175,949	53,416,615
Det minor	0.306	1.754	8.429	49.080	315.842	> 72gb	N/A
Gentleman	0.67	3.52	10.41	57.99	339.77	2058.20	N/A
Maple fac	1.91	3.48	23.11	57.75	509.82	7334.50	N/A
Maple tot	2.22	5.23	31.54	106.83	825.66	9392.70	-
Magma det	1.89	5.10	36.12	327.79	2108.42	> 72gb	N/A
Magma fac	1.21	7.58	158.97	583.39	13,640.79	> 72gb	N/A
Magma tot	3.10	12.68	195.09	911.18	15,749.21	-	-

Table 5.1: CPU timings in seconds for computing $\det(T_n)$ using Zippel's quadratic Vandermonde solver. N/A: Not attempted.

Step 10 evaluates the factors at Y_k to obtain univariate images. This step, which initially was a bottleneck, is optimized by taking advantage of previously computed evaluations. This optimization in the C code was done by Prof. Monagan.

For the bivariate Hensel lifts in step 12, I use the cubic algorithm of Monagan and Paluck [39, 49] that costs $O(d_1 d_j^2 + d_1^2 d_j)$ arithmetic operations in $\mathbb{Z}_p$. The C code is contributed by my colleague Garrett Paluck and I integrated it into my algorithm.

To solve the Vandermonde systems in step 18, a quadratic algorithm of Zippel [62] which does $O(s^2)$ arithmetic operations in $\mathbb{Z}_p$ has been implemented in C by Prof. Monagan. I integrated his C code into my algorithm in Maple. Prof. Monagan also contributed the Maple code for the fast Vandermonde solver by Kaltofen and Yagati [32], which costs $\mathcal{O}(s \log^2 s)$ arithmetic operations in $\mathbb{Z}_p$. I integrated the fast Vandermonde solver into my algorithm, and the first benchmark was significantly sped up.

For the monic and square-free case of algorithm CMBBSHL, I present two timing benchmarks. All timings were obtained on an Intel Xeon E5-2660 8 core CPU on the CECM server, `gaby.cecm.sfu.ca`. The first benchmark is for computing the factors of $\det(T_n)$, where T_n is a symmetric Toeplitz matrix. The old timings for which I used Zippel's quadratic Vandermonde solver [62] are shown in Tables 5.1 and 5.2. Table 5.1 shows the total CPU time in seconds for computing the factors of $\det(T_n)$. I compared our timings with timings for computing $\det(T_n)$ then factoring $\det(T_n)$ in Maple 2021 and in Magma V2.25−5. For constructing the black box for this benchmark, I implemented Bareiss' algorithm [1] in C to compute the determinant in $\mathbb{Z}_p$ in $O(n^2)$ arithmetic operations (see Section 7.3 for details) instead of Gaussian elimination, which costs $O(n^3)$.

For computations in Maple, Gentleman & Johnson's determinant algorithm [26] (coded by Prof. Monagan) is used to compute $\det(T_n)$. Maple 2021 uses MTSHL [47] for factoring polynomials.

In Table 5.1, n is the number of variables, the second row (CMBBSHL) is the total time for my new algorithm, and the third row is the total number of probes to the black box.

n	10	11	12	13	14	15	16
H.L. x_n total	1.045	1.819	9.256	20.785	143.883	266.496	4182.199
s (H.L. x_n)	522	814	3174	5223	19,960	34,081	127,690
BB	0.137	0.240	1.304	3.043	11.363	20.350	109.592
Interp2var	0.046	0.081	0.307	0.631	2.172	3.469	17.191
Eval $f_{\rho,j-1}$	0.153	0.262	1.327	2.931	21.158	41.056	683.224
BHL	0.106	0.180	0.754	1.678	5.200	8.238	51.347
VSolve	0.058	0.101	1.937	4.219	72.887	143.183	2903.867

Table 5.2: Breakdown of CPU timings in seconds for Hensel lifting the last variable x_n.

The second block shows the time for computing $\det(T_n)$ in the sparse representation using Maple's determinant command with the option `method=minor`, Prof. Monagan's implementation of the Gentleman-Johnson algorithm [26] and the time of Maple's `factor` command. The third block shows the time of determinant computation and factorization in Magma.

Table 5.2 is a summary of the breakdown of the four major sub-steps of Algorithm 11 for Hensel lifting the last variable x_n. The first block shows the total time for Hensel lifting the last variable x_n and the number of bivariate images needed to recover x_n which is s. In the second block, BB and Interp2var are timings for probes to the black box and bivariate dense interpolation to get $a_j(x_1, Y_k, x_j)$ (step 8). Eval $f_{\rho,j-1}$ is the time for evaluating $f_{\rho,j-1}$ (step 10). BHL is the time for bivariate Hensel lifts (step 12). VSolve is the time for solving Vandermonde systems (step 18).

My algorithm becomes faster than Maple at $n = 14$ and faster than Magma at $n = 12$. Maple runs out of memory (exceeds 72gigs) when trying to compute $\det(T_{16})$. Note that solving Vandermonde systems dominates the cost of my algorithm after $n = 13$. I further improved the timings by integrating a Maple implementation of the fast Vandermonde solver of Kaltofen and Yagati [32] which costs $\mathcal{O}(M(s)\log s)$. The new timings are shown in Tables 5.3 and 5.4. In [12], Connolly's experimentation with Monagan's implementation of the fast Vandermonde solver from [32] first beats Zippel's $O(s^2)$ Vandermonde solver at size $s = 2024$ (so at $n \geq 12$). We can see that in Table 5.3 that our new timings for CMBBSHL are faster than the old timings for $n \geq 12$. It only took 4876.8 seconds to compute the factors of $\det(T_{16})$, which is more than 3.5 times faster than using Zippel's quadratic Vandermonde solver [62]. The code for obtaining the new timings in Tables 5.3 and 5.4 is on the website `http://www.cecm.sfu.ca/~mmonagan/code/CMBBSHL/`.

In an earlier implementation of Kaltofen and Trager's algorithm using FOXBOX [16], it took 43 minutes to construct the black boxes of the factors of $\det(T_{16})$. However, this timing does not include the time for recovering the factors in the sparse representation using sparse polynomial interpolation [2, 62]. They have only been able to recover the factors of $\det(T_n)$ in the sparse representation up to $n = 13$, for which the sparse polynomial interpolation took 15 hours and 24 minutes. For their implementation, the processor was a Sun Untra 2 (168 MHz). Our computer is an Intel Xeon E5-2660 8 core CPU at 2.2/3.0 GHz (64 GB

n	10	11	12	13	14	15	16
CMBBSHL	6.299	14.679	43.927	106.838	403.089	1020.001	4876.827
# probes	109,139	267,465	894,358	2,180,399	6,981,462	17,175,949	53,416,615
Maple det	0.306	1.754	8.429	49.080	315.842	$>$ 72gb	N/A
Maple fac	1.91	3.48	23.11	57.75	509.82	7334.50	N/A
Maple tot	2.22	5.23	31.54	106.83	825.66	-	-

Table 5.3: CPU timings in seconds for computing $\det(T_n)$ using the fast Vandermonde solver. N/A: Not attempted.

n	10	11	12	13	14	15	16
H.L. x_n total	1.309	2.162	7.129	12.663	64.635	126.665	1041.959
t_n	522	814	3174	5223	19,960	34,081	127,690
BB	0.195	0.394	1.031	2.046	9.152	18.496	80.853
Interp2var	0.024	0.033	0.149	0.254	0.981	1.764	10.052
Eval $f_{i,j-1}$	0.061	0.099	0.634	1.269	14.709	32.935	508.658
BHL	0.578	0.992	3.455	6.234	24.352	45.136	240.095
VSolve	0.330	0.453	1.243	1.773	10.594	19.547	165.371

Table 5.4: Breakdown of CPU timings in seconds for Hensel lifting the last variable x_n.

RAM), which is more than 10 times faster. However, only 1 core was used for the benchmark timings. For our benchmark (Table 5.3), it took 4876.8 seconds ($\approx$ 81 minutes) in total to recover the factors for $n = 16$ in the sparse representation.

The first benchmark does not show the power of the black box approach, since $\det(T_n)$ has only two factors and they are relatively dense compared to the factors in the second benchmark. The second benchmark shows CPU times for factoring determinants which have four factors where the number of terms of the factors are much smaller than the number of terms of the determinant in the sparse representation. The matrices are generated as follows. First, two different $n \times n$ symmetric Toeplitz matrices T_1 and T_2 with multivariate polynomial entries are created. Then the $N \times N$ block diagonal matrix

$$B = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix}$$

is created. If we ask Maple to compute $\det B$, Maple's determinant routine will automatically identify the block structure and find the factorization $\det B = \det T_1 \det T_2$. To hide this factorization from Maple we create an upper triangular matrix P_u with diagonal entries 1 and entries above the diagonal chosen from $\{0, 1\}$ at random and also a lower triangular matrix P_l with diagonal entries 1 and entries below the diagonal chosen at random from $\{0, 1\}$. Now we create the input matrix $A = P_l B P_u$ so that

$$\det A = \det P_l \det B \det P_u = \det B = \det T_1 \det T_2.$$

For example, when $n = 2$, taking

$$T_1 = \begin{pmatrix} x_1 & x_2 + 5 \\ x_2 + 5 & x_1 \end{pmatrix} \text{ and } T_2 = \begin{pmatrix} x_1 & 2x_2^2 + 3 \\ 2x_2^2 + 3 & x_1 \end{pmatrix},$$

and matrices

$$P_u = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } P_l = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

we obtain the matrix

$$A = \begin{pmatrix} x_1 & x_2 + 5 & 2\,{x_2}^2 + x_2 + 8 & 2\,{x_2}^2 + 2\,x_1 + 3 \\ x_2 + 5 & x_1 & 2\,x_1 & 2\,{x_2}^2 + x_1 + x_2 + 8 \\ x_1 + x_2 + 5 & x_1 + x_2 + 5 & 5 + 2\,x_1 + x_2 & 2\,{x_2}^2 + 2\,x_1 + x_2 + 8 \\ x_1 & x_2 + 5 & 2\,{x_2}^2 + x_2 + 8 & 2\,{x_2}^2 + 2\,x_1 + 3 \end{pmatrix}.$$

We still have $\det(A) = \det(T_1)\det(T_2)$. However, Maple's determinant routine will no longer find this factorization.

The matrices for the second benchmark and Maple code and Magma code for computing and factoring their determinants can be found on the web under
`http://www.cecm.sfu.ca/~mmonagan/code/CMSHL/`.

The timings are shown in Tables 5.5 and 5.6. In both tables, the first block of rows are the number of variables n and the matrix size N. In Table 5.5, the second block shows the total time for algorithm CMBBSHL, the total number of probes to the black box, and information about the number of terms in each factor and the determinant $\det(A)$. Note that the number of terms in each factor is much smaller than $\#\det(A)$. The third block shows the time for computing $\det(A)$ using Prof. Monagan's implementation of the Gentleman & Johnson's algorithm, Maple's `factor` time and the total time Maple uses. The last block is the timings for Magma's determinant computation and factorization.

In this example, algorithm CMBBSHL is faster than Maple and Magma for all $n \geq 5$ and at $n = 8$, algorithm CMBBSHL is more than 330 times faster than Maple. Maple runs out of memory when computing $\det(A)$ at $n = 9$. The bottleneck is probes to the black box since, unlike the Toeplitz matrices, evaluations of the polynomial entries are needed prior to each determinant computation in $\mathbb{Z}_p$. In Table 5.6, the timings for probes to the black box is divided into BB eval and BB det. BB eval measures the time to evaluate the polynomial entries of the matrix A. BB det is the time for determinant computation in $\mathbb{Z}_p$.

n	5	6	7	8	9	10	11
$N = 2n$	10	12	14	16	18	20	22
CMBBSHL	0.383	1.537	4.778	20.971	77.894	342.264	1334.654
# probes	3064	10,772	27,490	95,212	278,098	973,240	3,089,700
$\#f_i$	12,24	52,63	69,147	319,363	411,891	1953,1951	2780,2634
	12,25	52,63	66,136	319,363	431,897	2066,2067	5768,6017
$\#\det(A)$	3644	19,750	70,522	811,363	3,980,956	36,906,753	147,531,107
Gentleman	0.675	13.717	161.22	6628.5	N/A	N/A	N/A
Maple fac	0.170	0.405	1.270	11.706	242.81	N/A	N/A
Maple tot	0.845	14.122	162.49	6640.206	-	-	-
Magma fac	0.030	1.810	13.020	1757.1	N/A	N/A	N/A
Magma det	1.600	20.490	422.77	>120,000	N/A	N/A	N/A
Magma tot	1.63	22.3	435.79	>121,757	-	-	-

Table 5.5: CPU timings for computing $\det(A)$ using Zippel's quadratic Vandermonde solver. N/A: Not attempted.

n	5	6	7	8	9	10	11
$N = 2n$	10	12	14	16	18	20	22
H.L. x_n total	0.086	0.116	0.273	0.997	2.675	9.690	31.161
s (H.L. x_n)	10	28	80	218	466	1221	3074
BB tot	0.029	0.039	0.116	0.514	1.580	6.362	21.586
BB eval	0.019	0.025	0.078	0.382	1.239	5.165	17.610
BB det	0.002	0.007	0.021	0.065	0.220	0.787	2.710
Interp2var	0.002	0.003	0.004	0.016	0.037	0.156	0.258
Eval $f_{\rho,j-1}$	0.004	0.023	0.045	0.126	0.275	0.898	2.385
BHL	0.002	0.006	0.013	0.041	0.088	0.233	0.596
VSolve	0.003	0.003	0.003	0.009	0.033	0.263	1.221

Table 5.6: Breakdown of CPU timings in seconds for Hensel lifting the last variable x_n.

Chapter 6

The complete black box factorization algorithm CMBBSHL

In this chapter, we present the complete factorization algorithm CMBBSHL that handles all cases of input, i.e. the non-monic, non-square-free, and non-primitive polynomial input $a \in \mathbb{Z}[x_1, \cdots, x_n]$. A variety of test examples with timing benchmarks are presented in Section 6.3. A detailed complexity analysis with failure probabilities is presented in Section 6.4. Finally, the case of large integer coefficients is also considered with timing benchmarks in Section 6.5.

The work in this chapter is new and the entire chapter is my own contribution.

6.1 Non-monic and non-square-free cases

If the input polynomial a is in the sparse representation and is non-monic, two methods are known to pre-compute the leading coefficients of the factors. One is Wang's *leading coefficient correction* [55], and the other is by Kaltofen [27]. However, for our black box factorization algorithm CMBBSHL, pre-computing the leading coefficients of the factors is not necessary and would compromise the complexity if $\mathrm{LC}(a, x_1)$ is large. In order to obtain the correct leading coefficients of the factors, I discovered a new strategy which is to scale the bivariate images at each Hensel lifting step to match their leading coefficients with the input factors. The details of this technique are explained in this section.

Similar to the monic and square-free case (the first four steps are identical), the following steps are performed prior to sparse Hensel lifting:

1. Choose a large prime p, e.g. $p = 2^{62} - 57$ and a positive integer $\tilde{N} < p$.
2. Choose an evaluation point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}^{n-1}$ randomly from $[1, \tilde{N} - 1]^{n-1}$. In my implementation, I choose $\tilde{N} = 4001$.
3. Compute $d_i = \deg(a, x_i)$ for all $1 \leq i \leq n$ w.h.p. (e.g. by Algorithm 4).

4. Compute $a(x_1, \boldsymbol{\alpha}) \bmod p$ w.h.p. by a univariate dense interpolation. Then $a(x_1, \boldsymbol{\alpha}) \in \mathbb{Z}[x_1]$ is computed using Chinese remaindering with different primes to recover the coefficients of $a(x_1, \boldsymbol{\alpha})$ in $\mathbb{Z}$.

5. $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}$. Let the irreducible factorization of a over $\mathbb{Z}$ be of the form

$$a = h f_1^{e_1} f_2^{e_2} \cdots f_r^{e_r} \in \mathbb{Z}[x_1, \cdots, x_n], \tag{6.1}$$

where $\deg(f_\rho, x_1) > 0$, f_ρ is irreducible over $\mathbb{Z}$ and primitive in x_1 with $\operatorname{sign}(f_\rho) = 1$ for $1 \le \rho \le r$, $h = \operatorname{cont}(a, x_1) \in \mathbb{Z}[x_2, \cdots, x_n]$ is the *generic content* (which is multiplied by $\operatorname{sign}(a)$, see Definition 1.4.7) of a in x_1 and it is not factored at this stage.

Let $\lambda_\rho := \operatorname{icont}(f_\rho(x_1, \boldsymbol{\alpha})) \in \mathbb{Z}$ for $1 \le \rho \le r$ and let $\lambda_h := \prod_{\rho=1}^r \lambda_\rho^{e_\rho}$. Let

$$\hat{f}_\rho := \frac{1}{\lambda_\rho} f_\rho(x_1, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_n] \text{ for } 1 \le \rho \le r \text{ and} \tag{6.2}$$

$$\hat{h} := \lambda_h h(x_2, \cdots, x_n) \in \mathbb{Z}[x_2, \cdots, x_n]. \tag{6.3}$$

With high probability (w.h.p.), $\boldsymbol{\alpha}$ is Hilbertian. Thus,

$$\begin{aligned} a(x_1, \boldsymbol{\alpha}) &= h(\boldsymbol{\alpha}) f_1(x_1, \boldsymbol{\alpha})^{e_1} \cdots f_r(x_1, \boldsymbol{\alpha})^{e_r} \\ &= h(\boldsymbol{\alpha}) \left(\lambda_1 \hat{f}_1(x_1, \boldsymbol{\alpha})\right)^{e_1} \cdots \left(\lambda_r \hat{f}_r(x_1, \boldsymbol{\alpha})\right)^{e_r} \text{ w.h.p.} \qquad (6.4) \\ &= \underbrace{h(\boldsymbol{\alpha}) \left(\prod_{\rho=1}^r \lambda_\rho^{e_\rho}\right)}_{\hat{h}(\boldsymbol{\alpha})} \hat{f}_1(x_1, \boldsymbol{\alpha})^{e_1} \cdots \hat{f}_r(x_1, \boldsymbol{\alpha})^{e_r}. \\ &= \hat{h}(\boldsymbol{\alpha}) \hat{f}_1(x_1, \boldsymbol{\alpha})^{e_1} \cdots \hat{f}_r(x_1, \boldsymbol{\alpha})^{e_r} \in \mathbb{Z}[x_1], \qquad (6.5) \end{aligned}$$

where $\hat{f}_\rho(x_1, \boldsymbol{\alpha})$ is primitive (i.e. $\operatorname{icont}(\hat{f}_\rho(x_1, \boldsymbol{\alpha})) = 1$ and $\operatorname{sign}(\hat{f}_\rho(x_1, \boldsymbol{\alpha})) = 1$) and irreducible over $\mathbb{Z}$.

We define the *square-free part* of the input polynomial a as (see also Section 3.4):

Definition 6.1.1.

$$\operatorname{sqf}(a) := \prod_{\rho=1}^r f_\rho \in \mathbb{Z}[x_1, \cdots, x_n]. \tag{6.6}$$

Note: $\operatorname{sqf}(a) = \pm a / \gcd(a, \partial a / \partial x_1)$ by Lemma 3.4.2.

Define $\Lambda := \prod_{\rho=1}^r \lambda_\rho$. By (6.2), w.h.p. (since $\boldsymbol{\alpha}$ is Hilbertian),

$$\operatorname{sqf}(a) = \prod_{\rho=1}^r \lambda_\rho \prod_{\rho=1}^r \hat{f}_\rho(x_1, \cdots, x_n) = \Lambda \prod_{\rho=1}^r \hat{f}_\rho(x_1, \cdots, x_n) \in \mathbb{Z}[x_1, \cdots, x_n], \tag{6.7}$$

where r is the number of factors in $\operatorname{sqf}(a)$, and $\hat{f}_\rho(x_1, \cdots, x_n) \in \mathbb{Q}[x_1, \cdots, x_n]$ if $|\lambda_\rho| > 1$.

Example 15. Let $a = hf_1f_2$ where $h = 2x_3$, $f_1 = 3x_1^2x_2^2 + x_1x_2x_3^2 + 2x_2x_3^4 + 3$ and $f_2 = 3x_1x_2 + 4x_3$. Let $\boldsymbol{\alpha} = (15, 7)$. Then,

$$\begin{aligned} a(x_1, \boldsymbol{\alpha}) &= h(\boldsymbol{\alpha}) f_1(x_1, \boldsymbol{\alpha}) f_2(x_1, \boldsymbol{\alpha}) \\ &= \underbrace{14}_{h(\boldsymbol{\alpha})} \underbrace{(3(225x_1^2 + 245x_1 + 24011))}_{f_1(x_1, \boldsymbol{\alpha})} \underbrace{(45x_1 + 28)}_{f_2(x_1, \boldsymbol{\alpha})} \\ &= \underbrace{42}_{\hat{h}(\boldsymbol{\alpha})} \underbrace{(225x_1^2 + 245x_1 + 24011)}_{\hat{f}_1(x_1, \boldsymbol{\alpha})} \underbrace{(45x_1 + 28)}_{\hat{f}_2(x_1, \boldsymbol{\alpha})}. \end{aligned}$$

In this example, $\lambda_1 = 3$, $\lambda_2 = 1$, and $\hat{h}(\boldsymbol{\alpha}) = (\lambda_1\lambda_2)h(\boldsymbol{\alpha}) = 3 \cdot 14 = 42$. In our algorithm the λ_ρ's are only determined after the last Hensel lifting step using rational number reconstruction.

Define $a_j := a(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ for $1 \leq j \leq n$ and $\hat{f}_{\rho,1} := \hat{f}_\rho(x_1, \boldsymbol{\alpha}) \bmod p$. Define $\hat{f}_{\rho,j} := \hat{f}_\rho(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ for $2 \leq j \leq n$ (to be computed).

The input and output to algorithm CMBBSHL (Algorithm 12) for the non-monic and non-square-free case is:

- Input: The modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$,
  $(\hat{f}_{\rho,1}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1]^r$, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, a prime p, $d_i = \deg(a, x_i)$ for $1 \leq i \leq n$ (pre-computed), $X = [x_1, \cdots, x_n]$, $n \in \mathbb{Z}$ s.t.
  (i) $\gcd(\hat{f}_{k,1}, \hat{f}_{l,1}) = 1$ for $k \neq l$ in $\mathbb{Z}_p[x_1]$,
  (ii) $\mathrm{sqf}(a_1) = \Lambda \prod_{\rho=1}^r \hat{f}_{\rho,1} \bmod p \in \mathbb{Z}_p[x_1]$, $\Lambda \in \mathbb{Z}_p$,
  (iii) $\hat{f}_{\rho,1}$ is primitive in x_1 for all $1 \leq \rho \leq r$.

- Output: $(\hat{f}_{\rho,n}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1, \cdots, x_n]^r$ s.t.
  (i) $\mathrm{sqf}(a_n) = \Lambda \prod_{\rho=1}^r \hat{f}_{\rho,n} \bmod p \in \mathbb{Z}_p[x_1, \cdots, x_n]$, $\Lambda \in \mathbb{Z}_p$,
  (ii) $\hat{f}_{\rho,n}(x_1, \boldsymbol{\alpha}) = \hat{f}_{\rho,1} \bmod p$ for all $1 \leq \rho \leq r$,
  (iii) $\hat{f}_{\rho,n}$ is primitive in x_1 for all $1 \leq \rho \leq r$;
  Or FAIL.

In our algorithm, $(\hat{f}_{\rho,1}, 1 \leq \rho \leq r)$ are irreducible in $\mathbb{Z}[x_1]$ but not necessarily irreducible in $\mathbb{Z}_p[x_1]$. Condition (i) of the input is needed for BHL. Our algorithm CMBBSHL could also lift the square-free factorization of $\mathrm{sqf}(a(x_1, \boldsymbol{\alpha}))$ if input condition (i) holds mod p.

Algorithm CMBBSHL lifts $(\hat{f}_{\rho,1}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1]^r$ to $(\hat{f}_{\rho,2}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1, x_2]^r$ then lifts $(\hat{f}_{\rho,2}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1, x_2]^r$ to $(\hat{f}_{\rho,3}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1, x_2, x_3]^r$ etc. After the j^{th} Hensel lifting step (see Algorithm 13), $\mathrm{sqf}(a_j) = \Lambda \prod_{\rho=1}^r \hat{f}_{\rho,j} \bmod p$ and $\hat{f}_{\rho,j}(x_j = \alpha_j) = \hat{f}_{\rho,j-1}$ for all $1 \leq \rho \leq r$. After the n^{th} step, $\mathrm{sqf}(a_n) = \Lambda \prod_{\rho=1}^r \hat{f}_{\rho,n}$. (A proof of correctness is given in Section 6.1.2.)

The j^{th} Hensel lifting step of algorithm CMBBSHL for the non-monic and non-square-free case is presented in Algorithm 13. Algorithm CMBBSHL could return FAIL at several

Algorithm 12 CMBBSHL: non-monic and non-square-free.

Input: The modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$, $(\hat{f}_{\rho,1}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1]^r$, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, a prime p, $d_i = \deg(a, x_i)$ for $1 \leq i \leq n$ (pre-computed), $X = [x_1, \cdots, x_n]$, $n \in \mathbb{Z}$ (the recursive variable) s.t. conditions (i)-(iii) of the input are satisfied.

Output: $(\hat{f}_{\rho,n}, 1 \leq \rho \leq r) \in \mathbb{Z}_p[x_1, \cdots, x_n]^r$ s.t. conditions (i)-(iii) of the output are satisfied. Otherwise, **return** FAIL.

1: **if** $n = 2$ **then**
2: $A_k \leftarrow a_2(x_1, x_2) \in \mathbb{Z}_p[x_1, x_2]$. // via probes to $\mathbf{B}$ and bivariate dense interpolation ..... $\mathcal{O}(d_1 d_2 \mathrm{C}(\text{probe } \mathbf{B})) + \mathcal{O}(d_1^2 d_2 + d_1 d_2^2)$
3: **if** $\deg(A_k, x_1) \neq d_1$ **or** $\deg(A_k, x_2) \neq d_2$ **then return** FAIL **end if**
4: $g_k \leftarrow \gcd(A_k, \frac{\partial A_k}{\partial x_1}) \bmod p \in \mathbb{Z}_p[x_1, x_2]$. ..... $\mathcal{O}(d_1^2 d_2 + d_1 d_2^2)$
5: **if** $\deg(g_k, x_1) \neq d_1 - \sum_{\rho=1}^{r} \deg(\hat{f}_{\rho,1}, x_1)$ **then return** FAIL **end if**
6: $A_{sf} \leftarrow \mathrm{quo}(A_k, g_k) \bmod p$. // $A_{sf} = \mathrm{sqf}(A_k) \bmod p$, up to a constant in $\mathbb{Z}_p$.
7: $A_{sfm} \leftarrow A_{sf}/(\mathrm{LC}(\mathrm{LC}(A_{sf}, x_1), x_2)) \bmod p$. // make $\mathrm{LC}(A_{sf}, x_1)$ monic in x_2.
8: **return** BivariateHenselLift$(A_{sfm}, (\hat{f}_{\rho,1}, 1 \leq \rho \leq r), \alpha_2, p)$ ..... $\mathcal{O}(d_1^2 d_2 + d_1 d_2^2)$
9: **end if**
10: $(\hat{f}_{\rho,n-1}, 1 \leq \rho \leq r) \leftarrow$ CMBBSHL$(\mathbf{B}, (\hat{f}_{\rho,1}, 1 \leq \rho \leq r), \boldsymbol{\alpha}, p, d_i, X, n-1)$.
11: **return** CMBBSHLstepj$(\mathbf{B}, (\hat{f}_{\rho,n-1}, 1 \leq \rho \leq r), \boldsymbol{\alpha}, p, d_i, X, n)$ // Algorithm 13

steps, i.e. at step 4, 9, 11,15,16, or 29 of Algorithm 13. If it returns FAIL at step 29, then either the weak SHL assumption fails or the initial evaluation point $\boldsymbol{\alpha}$ is not Hilbertian. Algorithm CMBBSHL could also return an incorrect answer with a low probability. If p divides any integer coefficient of any irreducible factor $f_\rho \in \mathbb{Z}[x_1, \cdots, x_n]$, then algorithm CMBBSHL returns an answer that is not FAIL but is incorrect. We present a detailed failure probability analysis in Section 6.4.

The key idea of Algorithm CMBBSHL is to interpolate the square-free part of the bivariate images of a and then use them to perform non-monic bivariate Hensel lifts. In steps 10–12 of Algorithm 13, a square-free image of a, $\mathrm{sqf}((a_j(x_1, Y_k, x_j))$, is computed via a bivariate gcd computation and a division, i.e.

$$A_{sf} := \frac{a_j(x_1, Y_j, x_j)}{\gcd\left(a_j(x_1, Y_k, x_j), \frac{\partial}{\partial x_1}\left(a_j(x_1, Y_k, x_j)\right)\right)}.$$

$A_{sf} = \mathrm{sqf}(a_j(x_1, Y_j, x_j))$ up to a scalar in $\mathbb{Z}_p$. This removes the content of $a_j(x_1, Y_k, x_j)$ in x_1, a polynomial in x_j, by Lemma 3.4.2.

Step 13 then makes $\mathrm{LC}(A_{sf}, x_1)$ monic, i.e. $\mathrm{LC}(\mathrm{sqf}(a_j(x_1, Y_k, x_j)), x_1)$ becomes monic in x_j. Step 14 evaluates the input factors to get a univariate image $\hat{f}_{\rho,j-1}(x_1, Y_k)$. Then at step 17, a non-monic bivariate Hensel lift (BHL) is performed, so we get a bivariate image of the factors, $\hat{f}_{\rho,j}(x_1, Y_k, x_j)$. After obtaining s bivariate images of the factors (s is defined in step 5 in Algorithm 13), we use them to recover the variables $x_2, \cdots, x_{j-1}$ in the factor $\hat{f}_{\rho,j} \in \mathbb{Z}_p[x_1, \cdots, x_j]$ by solving Vandermonde systems at step 23.

Algorithm 13 CMBBSHLstepj: Hensel lifting x_j (non-monic and non-square-free).

Input: The modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$, $(\hat{f}_{\rho,j-1}, 1 \le \rho \le r) \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]^r$, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, a prime p, $d_i = \deg(a, x_i)$ for $1 \le i \le n$ (pre-computed), $X = [x_1, \cdots, x_n]$, $j \in \mathbb{Z}$ s.t. $\text{sqf}(a_j(x_j = \alpha_j)) = \prod_{\rho=1}^r \lambda_\rho \prod_{\rho=1}^r \hat{f}_{\rho,j-1}$.

Output: $(\hat{f}_{\rho,j}, 1 \le \rho \le r) \in \mathbb{Z}_p[x_1, \cdots, x_j]^r$ s.t. (i) $\text{sqf}(a_j) = \prod_{\rho=1}^r \lambda_\rho \prod_{\rho=1}^r \hat{f}_{\rho,j}$, (ii) $\hat{f}_{\rho,j}(x_j = \alpha_j) = \hat{f}_{\rho,j-1}$ for all $1 \le \rho \le r$; Otherwise, **return** FAIL.

Let $\hat{f}_{\rho,j-1} = \sum_{i=0}^{df_\rho} \sigma_{\rho,i}(x_2, ..., x_{j-1}) x_1^i$ $(1 \le \rho \le r)$ where $\sigma_{\rho,i} = \sum_{k=1}^{s_{\rho,i}} c_{\rho,ik} M_{\rho,ik}$ with $M_{\rho,ik}$ the monomials in $\sigma_{\rho,i}$ and $df_\rho = \deg(\hat{f}_{\rho,j-1}, x_1)$.
Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-2}$ at random.
Evaluate (for $1 \le \rho \le r$): $\mathcal{S}_\rho = \{\mathcal{S}_{\rho,i} = \{m_{\rho,ik} = M_{\rho,ik}(\boldsymbol{\beta}), 1 \le k \le s_{\rho,i}\}, 0 \le i \le df_\rho\}$.
if any $|\mathcal{S}_{\rho,i}| \ne s_{\rho,i}$ **then return** FAIL **end if** // monomial evals must be distinct
Let s be the maximum of $s_{\rho,i}$.
// Compute s images of the factors in $\mathbb{Z}_p[x_1, x_j]$:
for k from 1 to s **do**
  Let $Y_k = (x_2 = \beta_2^k, \cdots, x_{j-1} = \beta_{j-1}^k)$.
  $A_k \leftarrow a_j(x_1, Y_k, x_j) \in \mathbb{Z}_p[x_1, x_j]$. // via probes to **B** and bivariate dense interpolation ..... $\mathcal{O}(sd_1d_j\text{C(probe }\mathbf{B})) + \mathcal{O}(s(d_1^2d_j + d_1d_j^2))$
  if $\deg(A_k, x_1) \ne d_1$ **or** $\deg(A_k, x_j) \ne d_j$ **then return** FAIL **end if**
  $g_k \leftarrow \gcd(A_k, \frac{\partial A_k}{\partial x_1}) \bmod p \in \mathbb{Z}_p[x_1, x_j]$. ..... $\mathcal{O}(s(d_1^2d_j + d_1d_j^2))$
  if $\deg(g_k, x_1) \ne d_1 - \sum_{\rho=1}^r df_\rho$ **then return** FAIL **end if**
  $A_{sf} \leftarrow \text{quo}\ (A_k, g_k) \bmod p$. // $A_{sf} = \text{sqf}(A_k) \bmod p$, up to a constant in $\mathbb{Z}_p$.
  $A_{sfm} \leftarrow A_{sf}/(\text{LC}(\text{LC}(A_{sf}, x_1), x_j)) \bmod p$. // make $\text{LC}(A_{sf}, x_1)$ monic in x_j.
  $F_{\rho,k} \leftarrow \hat{f}_{\rho,j-1}(x_1, Y_k) \in \mathbb{Z}_p[x_1]$ for $1 \le \rho \le r$. ..... $\mathcal{O}(s(\sum_{\rho=1}^r \#\hat{f}_{\rho,j-1}))$
  if any $\deg(F_{\rho,k}) < df_\rho$ (for $1 \le \rho \le r$) **then return** FAIL **end if**
  if $\gcd(F_{\rho,k}, F_{\phi,k}) \ne 1$ for any $1 \le \rho < \phi \le r$ **then return** FAIL **end if**
  $\hat{f}_{\rho,k} \leftarrow \text{BivariateHenselLift}(A_{sfm}(x_1, x_j), F_{\rho,k}(x_1), \alpha_j, p)$. ..... $\mathcal{O}(s(\tilde{d}_1\tilde{d}_j^2 + \tilde{d}_1^2\tilde{d}_j))$
end for
Let $\hat{f}_{\rho,k} = \sum_{l=1}^{t_\rho} \alpha_{\rho,kl} \tilde{M}_{\rho,l}(x_1, x_j) \in \mathbb{Z}_p[x_1, x_j]$ for $1 \le k \le s$, for $1 \le \rho \le r$ $(t_\rho = \#\hat{f}_{\rho,k})$.
for ρ from 1 to r **do**
  for l from 1 to t_ρ **do**
    $i \leftarrow \deg(\tilde{M}_{\rho,l}, x_1)$.
    Solve the linear system $\left\{\sum_{k=1}^{s_{\rho,i}} m_{\rho,ik}^t c_{\rho,lk} = \alpha_{\rho,tl} \text{ for } 1 \le t \le s_{\rho,i}\right\}$ for $c_{\rho,lk}$.
  end for ..... $\mathcal{O}(s\tilde{d}_j(\sum_{\rho=1}^r \#\hat{f}_{\rho,j-1}))$
  $\hat{f}_{\rho,j} \leftarrow \sum_{l=1}^{t_\rho} \left(\sum_{k=1}^{s_{\rho,i}} c_{\rho,lk} M_{\rho,ik}(x_2, ..., x_{j-1})\right) \tilde{M}_{\rho,l}(x_1, x_j)$.
end for
Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_j) \in \mathbb{Z}_p^{j-1}$ at random until $\deg(\hat{f}_{\rho,j}(x_1, \boldsymbol{\beta})) = df_\rho$ for all $1 \le \rho \le r$.
$A_{\boldsymbol{\beta}} \leftarrow a_j(x_1, \boldsymbol{\beta}) \bmod p$ via probes to **B** and Lagrange interpolation.
if $\hat{f}_{\rho,j}(x_1, \boldsymbol{\beta}) \mid A_{\boldsymbol{\beta}}$ for all $1 \le \rho \le r$ **then return** $(\hat{f}_{\rho,j}, 1 \le \rho \le r)$ **else return** FAIL
end if

After the last Hensel lifting step, rational number reconstruction (see also Section 3.7) is performed on the coefficients of $\hat{f}_{\rho,n}$ in $\mathbb{Z}_p$ for $1 \leq \rho \leq r$ to get the integer coefficients of the factors f_ρ in $\mathbb{Z}$. We elaborate. Suppose

$$f_\rho = \sum_{k=1}^{\#f_\rho} c_k M_k \in \mathbb{Z}[x_1, \cdots, x_n],$$

where $c_k \in \mathbb{Z}$ (to be determined), M_k is a monomial of f_ρ and $\#f_\rho$ is the number of terms in f_ρ. After the last Hensel lifting step, we have computed $\hat{f}_{\rho,n} = \sum_{k=1}^{\#f_\rho} \hat{c}_k M_k \in \mathbb{Z}_p[x_1, \cdots, x_n]$. Thus,

$$\hat{f}_{\rho,n} = \sum_{k=1}^{\#f_\rho} \hat{c}_k M_k \equiv \frac{1}{\lambda_\rho} f_\rho \bmod p \equiv \sum_{k=1}^{\#f_\rho} \frac{c_k}{\lambda_\rho} M_k \bmod p.$$

Thus, we use rational number reconstruction to obtain λ_ρ and c_k from $\hat{c}_k \equiv \frac{c_k}{\lambda_\rho} \bmod p$ $(1 \leq k \leq \#f_\rho)$. Then fractions are cleared and we obtain $f_\rho \in \mathbb{Z}[x_1, \cdots, x_n]$.

Remark 5. Rational number reconstruction may fail. If we use Wang's rational reconstruction [57], to guarantee the correct answer, we need

$$p > 2 \max\left(\max_{\rho=1}^{r} |\lambda_\rho|, \max_{\rho=1}^{r} \|f_\rho\|_\infty\right)^2, \tag{6.8}$$

where $\|f_\rho\|_\infty$ is the max-norm of the irreducible factor $f_\rho \in \mathbb{Z}[x_1, \cdots, x_n]$.

Example 16. Consider $a = f_1 f_2 \in \mathbb{Z}[x_1, \cdots, x_4]$ where

$$\begin{aligned} f_1 &= (2x_2^2x_3^3 + 4)x_1^8 + (4x_2^2x_3^3 + 22x_2^2x_4^3 + 1452x_2^2x_4)x_1 + x_2^2x_3x_4 - 4x_3, \\ f_2 &= (3x_2 + 39x_4 + 3x_3)x_1^8 + (5x_2x_3^2x_4 + 33x_2x_3x_4^2)x_1^2 - 363x_4^2 + 44. \end{aligned}$$

In this case, $h = 1$ (a has no content in x_1 and neither integer content) and $\mathrm{sqf}(a) = a$. Let $\boldsymbol{\alpha} = (2, 3, 9)$,

$$\begin{aligned} a(x_1, \boldsymbol{\alpha}) &= 80520x_1^{16} + 3706560x_1^{10} + \cdots - 3430775304x_1 - 2818464 \\ &= \underbrace{4}_{\lambda_1} \underbrace{(55x_1^8 + 29214x_1 + 24)}_{\hat{f}_1(x_1, \boldsymbol{\alpha})} \underbrace{(366x_1^8 + 16848x_1^2 - 29359)}_{\hat{f}_2(x_2, \boldsymbol{\alpha})} \\ &= f_1(x_1, \boldsymbol{\alpha}) f_2(x_1, \boldsymbol{\alpha}). \end{aligned}$$

We have $\lambda_1 = 4$ and $\lambda_2 = 1$, thus $f_1(x_1, \boldsymbol{\alpha}) = \lambda_1 \hat{f}_1 = 4\hat{f}_1$ and $f_2(x_1, \boldsymbol{\alpha}) = \hat{f}_2$. The input to algorithm CMBBSHL is $p = 2^{31} - 1$, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, the modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\beta}, p) = a(\boldsymbol{\beta}) \bmod p$, $\hat{f}_{\rho,1} = \hat{f}_\rho \bmod p$ $(\rho = 1, 2)$, and the pre-computed degrees $d_j = \deg(a, x_j) = [16, 3, 5, 5]$ for $j = 1, \cdots, 4$.

After the 1st (denoted as the 2nd) Hensel lifting step (which does a bivariate Hensel lift only), the algorithm outputs $\hat{f}_{\rho,2} \in \mathbb{Z}_p[x_1, x_2]$ (for $\rho = 1, 2$) such that $a_2 = \text{sqf}(a_2) = (\lambda_1\lambda_2)\hat{f}_{1,2}\hat{f}_{2,2} \bmod p$ with

$$\begin{aligned}\hat{f}_{1,2} &= (1073741837x_2^2 + 1)x_1^8 + 1073749127x_2^2x_1 + 1610612742x_2^2 + 2147483644,\\ \hat{f}_{2,2} &= (3x_2 + 360)x_1^8 + 8424x_2x_1^2 + 2147454288.\end{aligned}$$

After the 3rd Hensel lifting step, Algorithm 13 outputs

$$\begin{aligned}\hat{f}_{1,3} &= (1073741824x_2^2x_3^3 + 1)x_1^8 + (x_2^2x_3^3 + 1073749100x_2^2)x_1\\ &\quad + 536870914x_2^2x_3 + 2147483646x_3,\\ \hat{f}_{2,3} &= (3x_2 + 3x_3 + 351)x_1^8 + (45x_2x_3^2 + 2673x_2x_3)x_1^2 + 2147454288.\end{aligned}$$

The last Hensel lifting step outputs $\hat{f}_{\rho,4} \in \mathbb{Z}_p[x_1, x_2, x_3, x_4]$ ($\rho = 1, 2$) s.t. $a_4 = \text{sqf}(a_4) = (\lambda_1\lambda_2)\hat{f}_{1,4}\hat{f}_{2,4} \bmod p$ with

$$\begin{aligned}\hat{f}_{1,4} &= (1073741824x_2^2x_3^3 + 1)x_1^8 + (x_2^2x_3^3 + 1073741829x_2^2x_4^3 + 363x_2^2x_4)x_1\\ &\quad + 536870912x_2^2x_3x_4 + 2147483646x_3\\ \hat{f}_{2,4} &= (3x_2 + 39x_4 + 3x_3)x_1^8 + (5x_2x_3^2x_4 + 33x_2x_3x_4^2)x_1^2 + 2147483284x_4^2 + 44.\end{aligned}$$

Now, we notice that $4\hat{f}_{1,4} \bmod p = f_1$ and $\hat{f}_{2,4} \bmod p = f_2$ (where mod is taken in the symmetric range). The values for λ_1, λ_2 are still unknown, so we perform rational number reconstruction on the coefficients of $\hat{f}_{\rho,4}$ to find λ_ρ and hence get the true factors f_ρ ($\rho = 1, 2$). In Maple, we do the following for the first factor:

```
> fr[1] :=  iratrecon(f_hat[1,4],p);
```

$$fr_1 := \frac{1}{2}x_2^2x_1^8x_3^3 + x_1^8 + x_1x_2^2x_3^3 + \frac{11}{2}x_2^2x_1x_4^3 + 363x_2^2x_1x_4 + \frac{1}{4}x_2^2x_3x_4 - x_3$$

Observe that $\lambda_1 = 4$ is the least common multiple of the denominators of coefficients of fr_1. Multiply fr_1 by λ_1, we get the true factor $f_1 \in \mathbb{Z}[x_1, \cdots, x_n]$:

```
> f[1] := 4*fr[1];
```

$$f_1 := 2x_2^2x_1^8x_3^3 + 4x_1^8 + 4x_1x_2^2x_3^3 + 22x_2^2x_1x_4^3 + 1452x_2^2x_1x_4 + x_2^2x_3x_4 - 4x_3$$

Similarly for the second factor f_2:

```
> fr[2] :=  iratrecon(f_hat[2,4],p);
```

$$fr_2 := 3x_1^8x_2 + 3x_1^8x_3 + 39x_1^8x_4 + 5x_1^2x_2x_3^2x_4 + 33x_1^2x_2x_3x_4^2 - 363x_4^2 + 44$$

There is no fraction in fr_2, so $f_2 = fr_2$.

6.1.1 Non-monic bivariate Hensel lifting

Define $lc_{Asf} := \text{LC}(\text{LC}(\text{sqf}(a_j(x_1, Y_k, x_j), x_1), x_j)$. The non-monic bivariate Hensel lift (step 17 of Algorithm 13) has the following input and output:

- Input: A prime p, $\alpha_j \in \mathbb{Z}_p$, $A_{sfm} = \text{monic}(\text{sqf}(a_j(x_1, Y_k, x_j))$ (step 13), $F_{\rho,k} = \hat{f}_{\rho,j-1}(x_1, Y_k)$ $(1 \leq \rho \leq r)$ (step 14) s.t.
  (i) $\gcd(\hat{f}_{\rho,j-1}(x_1, Y_k), \hat{f}_{\phi,j-1}(x_1, Y_k)) = 1$ for $\rho \neq \phi$,
  (ii) $A_{sfm}(x_j = \alpha_j) = \xi \prod_{\rho=1}^{r} \hat{f}_{\rho,j-1}(x_1, Y_k),\ \xi \in \mathbb{Z}_p$,
  (iii) $F_{\rho,k} = \hat{f}_{\rho,j-1}(x_1, Y_k)$ is primitive in x_1 for all $1 \leq \rho \leq r$.

- Output: $\hat{f}_{\rho,j}(x_1, Y_k, x_j)$ $(1 \leq \rho \leq r)$ s.t.
  (i) $A_{sfm} = \xi \prod_{\rho=1}^{r} \hat{f}_{\rho,j}(x_1, Y_k, x_j),\ \xi \in \mathbb{Z}_p$,
  (ii) $\hat{f}_{\rho,j}(x_1, Y_k, x_j)(x_j = \alpha_j) = \hat{f}_{\rho,j-1}(x_1, Y_k)$ for all $1 \leq \rho \leq r$,
  (iii) $\hat{f}_{\rho,j}(x_1, Y_k, x_j)$ is primitive in x_1 for all $1 \leq \rho \leq r$;
  Or FAIL.

In the above, monic(.) means $\text{LC}(\text{sqf}(a_j(x_1, Y_k, x_j)), x_1)$ is monic in x_j. Also, we have $\xi = \Lambda / lc_{Asf} \in \mathbb{Z}_p$.

If Algorithm 14 returns FAIL, it means the factorization $A_{sfm} = \xi \prod_{\rho=1}^{r} \hat{f}_{\rho,j}(x_1, Y_k, x_j)$ does not exist. This could imply that the initial evaluation point $\boldsymbol{\alpha}$ is not Hilbertian.

Notice that the output of BHL $\hat{f}_{\rho,j}(x_1, Y_k, x_j)$ satisfies

$$\hat{f}_{\rho,j}(x_1, Y_k, x_j)(x_j = \alpha_j) = \hat{f}_{\rho,j-1}(x_1, Y_k)\ (1 \leq \rho \leq r).$$

Thus, when evaluating the output bivariate factors $\hat{f}_{\rho,j}(x_1, Y_k, x_j)$ at $x_j = \alpha_j$, their leading coefficients equal the leading coefficients of the input factors $\hat{f}_{\rho,j-1}(x_1, Y_k)$. The correct return of the leading coefficients from BHL ensures that we have the correct leading coefficients after each Hensel lifting step of CMBBSHL, i.e. $\hat{f}_{\rho,j} \in \mathbb{Z}_p[x_1, \cdots, x_j]$ satisfies

$$\hat{f}_{\rho,j}(x_j = \alpha_j) = \hat{f}_{\rho,j-1}\ (1 \leq \rho \leq r).$$

We modified the monic BHL algorithm developed by Monagan and Paluck in 2022 [48] to handle the non-monic case. It has a cubic cost of $O(d_1 d_j^2 + d_1^2 d_j)$. Pseudocode for non-monic bivariate Hensel lifting is shown in Algorithm 14.

In Algorithm 14, there is a potential issue when $\gamma(y) = \text{LC}(a, x)$ has a high degree. Thus after step 2 of Algorithm 14, $a(x, y) \leftarrow \gamma(y)^{r-1} a(x, y)$ has a high degree in y. This may happen if the number of factors is large. In such case, we could implement a recursive algorithm to break down the factors into a binary tree for bivariate Hensel lifts.

6.1.2 Proof of correctness

We need Proposition 6.1.2 to prove Theorem 6.1.3.

Algorithm 14 Non-monic bivariate Hensel lifting - cubic cost.

Input: A prime p, $\alpha \in \mathbb{Z}_p$, $a \in \mathbb{Z}_p[x,y]$ where a is primitive in x and $\mathrm{LC}(\mathrm{LC}(\mathsf{a},\mathsf{x}),\mathsf{y}) = 1$, $f_{\rho,0} \in \mathbb{Z}_p[x]$ for $1 \le \rho \le r$ s.t.
(i) $\gcd(f_{k,0}, f_{l,0}) = 1$ for $k \neq l$,
(ii) $a(y=\alpha) = \xi \prod_{\rho=1}^{r} f_{\rho,0},\ \xi \in \mathbb{Z}_p$,
(iii) $f_{\rho,0}$ is primitive in x for all $1 \le \rho \le r$.
Output: $f_\rho \in \mathbb{Z}_p[x,y]$ for $1 \le \rho \le r$ s.t.
(i) $a = \xi \prod_{\rho=1}^{r} f_\rho$,
(ii) $f_\rho(y=\alpha) = f_{\rho,0}$,
(iii) f_ρ is primitive in x for all $1 \le \rho \le r$; Or FAIL.

$\gamma(y) \leftarrow lc(a,x) \in \mathbb{Z}_p[y]$;
$a(x,y) \leftarrow \gamma(y)^{r-1} a(x,y) \in \mathbb{Z}_p[x,y]$.
$f_{\rho,0} \leftarrow \gamma(y) \cdot \mathrm{monic}(f_{\rho,0}(x)) \bmod (y-\alpha) \in \mathbb{Z}_p[x]$ $(1 \le \rho \le r)$.
$dx \leftarrow \deg(a,x)$; $dy \leftarrow \deg(a,y)$; $df_{\rho,0} \leftarrow \deg(f_{\rho,0},x)$.
$M \leftarrow \prod_{\rho=1}^{r} f_{\rho,0} \in \mathbb{Z}_p[x]$.
for ρ from 1 to r **do** $f_\rho \leftarrow f_{\rho,0}$; $M_\rho \leftarrow M/f_{\rho,0}$ **end for**
for k from 0 to dy **do**
  $\gamma_k \leftarrow \mathrm{coeff}(\gamma, (y-\alpha)^k)$.
  for ρ from 1 to r **do** $Tf_{\rho,k} \leftarrow \gamma_k x^{df_{\rho,0}}$ **end for**
end for
for k from 1 to dy **do**
  $ac_k \leftarrow \mathrm{coeff}(a, (y-\alpha)^k)$.
  $\Delta_k \leftarrow \mathrm{coeff}(\prod_{\rho=1}^{r} f_\rho, (y-\alpha)^k)$. // via eval and interpolation.
  $\delta_k \leftarrow \sum_{\rho=1}^{r} Tf_{\rho,k} \cdot M_\rho$.
  $c_k \leftarrow ac_k - \Delta_k - \delta_k$.
  if $\sum_{\rho=1}^{r} \deg(f_\rho, y) = dy$ and $c_k \neq 0$ **return FAIL end if**
  if $c_k \neq 0$ **then**
    Solve $\sum_{\rho=1}^{r} \bar{f}_{\rho,k} M_\rho = c_k$ for $\bar{f}_{\rho,k} \in \mathbb{Z}_p[x]$ with $\deg(\bar{f}_{\rho,k}, x) < \deg(f_{\rho,0}, x)$ $(1 \le \rho \le r)$.
    for ρ from 1 to r **do**
      $f_{\rho,k} \leftarrow Tf_{\rho,k} + \bar{f}_{\rho,k}$; $f_\rho \leftarrow f_\rho + f_{\rho,k}(y-\alpha)^k$.
    end for
  end if
end for
if $\sum_{\rho=1}^{r} \deg(f_\rho, y) \neq dy$ **then return FAIL end if**
if $c_k = 0$ **then**
  for ρ from 1 to r **do**
    $\tilde{f}_\rho \leftarrow \mathrm{primpart}(f_\rho(x,y), x))$. // $\mathrm{lc}(\mathrm{lc}(\tilde{f}_\rho, x), y) = 1$.
    $lc_{eval} \leftarrow \mathrm{lc}(\tilde{f}_\rho(x,y), x)(y=\alpha)$; $\eta \leftarrow \mathrm{lc}(f_{\rho,0}, x)/lc_{eval}$.
    $f_\rho \leftarrow \eta \tilde{f}_\rho$.
  end for
  return f_ρ for $1 \le \rho \le r$.
else
  return FAIL
end if

Proposition 6.1.2. *The output factors $f_\rho \in \mathbb{Z}_p[x,y]$ $(1 \le \rho \le r)$ from Algorithm 14 (non-monic BHL) are uniquely determined.*

Proof. Since $\deg(\bar{f}_{\rho,k}, x) < \deg(f_{\rho,0}, x)$, the multi-term Diophantine equation (step 18) gives unique solutions $\bar{f}_{\rho,k}$ (Theorem 2.6 in [19]). By construction in step 20, $f_{\rho,k} = \gamma_k x^{df_{\rho,0}} + \bar{f}_{\rho,k}$, where $\gamma_k x^{df_{\rho,0}}$ is the leading term of $f_{\rho,k}$. Since $\deg(\bar{f}_{\rho,k}, x) < \deg(f_{\rho,0}, x)$ in step 18, the leading coefficients of $f_{\rho,k}$ do not change, and the solution $f_\rho^{(k+1)} = f_{\rho,0} + f_{\rho,1}(y-\alpha) + \cdots + f_{\rho,k}(y-\alpha)^k$ obtained at the k^{th} iteration (step 20) is uniquely determined.

In Algorithm 14, the input polynomial a is A_{sfm} which comes from step 13 of Algorithm 13. Since $\mathrm{LC}(\mathrm{LC}(A_{sfm}, x), y) = 1$, for each $\tilde{f}_\rho = \mathrm{primpart}(f_\rho(x,y), x))$, $\mathrm{LC}(\mathrm{LC}(\tilde{f}_\rho, x), y) = 1$ and the $\tilde{f}_\rho$'s $(1 \le \rho \le r)$ are also uniquely determined. This means

$$a = \prod_{\rho=1}^{r} \tilde{f}_\rho \in \mathbb{Z}_p[x,y]. \tag{6.9}$$

By evaluating (6.9) at $y = \alpha$, we get

$$a|_{y=\alpha} = \tilde{f}_1 \tilde{f}_2 \cdots \tilde{f}_r|_{y=\alpha} = \xi f_{1,0} f_{2,0} \cdots f_{r,0} \in \mathbb{Z}_p[x],$$

where $\xi \in \mathbb{Z}_p$. Since $\mathbb{Z}_p$ is a field, there exists $\eta_\rho \in \mathbb{Z}_p$ s.t. $\tilde{f}_\rho(y = \alpha) = (1/\eta_\rho) f_{\rho,0}$ for all $1 \le \rho \le r$.
Define $f_\rho := \eta_\rho \tilde{f}_\rho(x,y)$ $(1 \le \rho \le r)$. Since $\mathbb{Z}_p[x,y]$ is a UFD,

$$a = \prod_{\rho=1}^{r} \tilde{f}_\rho = \frac{\prod_{\rho=1}^{r} f_\rho}{\prod_{\rho=1}^{r} \eta_\rho} = \xi \prod_{\rho=1}^{r} f_\rho \in \mathbb{Z}_p[x,y],$$

where $\xi = 1/(\prod_{\rho=1}^{r} \eta_\rho)$, and f_ρ's are uniquely determined. □

Theorem 6.1.3. *Let* $\mathrm{sqf}(a_j) = \mathrm{sqf}(a(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n))$ *mod* $p \in \mathbb{Z}_p[x_1, \cdots, x_j]$. *Let r be the number of square-free factors of $a \in \mathbb{Z}[x_1, \cdots, x_n]$ and $\Lambda = \prod_{\rho=1}^{r} \lambda_\rho \in \mathbb{Z}$ where $\lambda_\rho = \mathrm{icont}(f_\rho(x_1, \boldsymbol{\alpha}))$. Let $\hat{f}_{\rho,j} \in \mathbb{Z}_p[x_1, \cdots, x_j]$ be the output factors after the j^{th} Hensel lifting step of CMBBSHL (Algorithm 13). Then,*

$$\mathrm{sqf}(a_j) = \Lambda \prod_{\rho=1}^{r} \hat{f}_{\rho,j} \ \textit{mod}\ p, \tag{6.10}$$

for all $1 \le j \le n$.

Proof. We want to show that (6.10) is satisfied for each j $(1 \le j \le n)$. The first Hensel lifting step $(j = 1)$ is the initial input, and (6.10) is satisfied. For $j = 2$, it is a bivariate Hensel lift, and from Proposition 6.1.2, (6.10) is satisfied.

Suppose (6.10) is satisfied at the beginning of Hensel lifting step j, i.e. (6.10) is satisfied for $j-1$. Before each bivariate Hensel lift,

$$\begin{aligned} \text{sqf}(a_j(x_1, Y_k, \alpha_j)) &= \Lambda \prod \hat{f}_{\rho,j-1}(x_1, Y_k) \bmod p \\ \Rightarrow \text{monic}(\text{sqf}(a_j(x_1, Y_k, \alpha_j))) &= \Lambda / lc_{Asf} \prod \hat{f}_{\rho,j-1}(x_1, Y_k) \bmod p. \end{aligned}$$

And after each BHL, Proposition 6.1.2 ensures unique $\hat{f}_{\rho_j}$'s s.t.

$$\begin{aligned} \text{monic}(\text{sqf}(a_j(x_1, Y_k, x_j))) &= \Lambda / lc_{Asf} \prod \hat{f}_{\rho,j}(x_1, Y_k, x_j) \bmod p \\ \Rightarrow \text{sqf}(a_j(x_1, Y_k, x_j)) &= \Lambda \prod \hat{f}_{\rho,j}(x_1, Y_k, x_j) \bmod p. \end{aligned}$$

After all bivariate Hensel lifts, the Vandermonde solves give unique solutions for coefficients $c_{\rho,lk}$ (since the monomial evaluations are distinct in step 3, $\det(V) \neq 0$, see Section 3.6.2 for details) to recover $\hat{f}_{\rho,j}$. Therefore,

$$\text{sqf}(a_j) = \Lambda \prod_{\rho=1}^{r} \hat{f}_{\rho,j} \bmod p.$$

□

6.2 Non-primitive case: content computation

Algorithm CMBBSHL returns the irreducible factors of the primitive part of a for $a \in \mathbb{Z}[x_1, \cdots, x_n]$ (w.h.p.) in their sparse representation. In order to compute the irreducible factors of the content of a, we construct another black box for the content of a and compute its irreducible factors recursively.

Let $a = h \prod_{\rho=1}^{r} f_\rho^{e_\rho}$, where $h = \text{cont}(a, x_1)$ and f_ρ's are the irreducible factors of $\text{pp}(a)$ which have been computed already by Algorithm CMBBSHL (Algorithm 12). To compute $h(\boldsymbol{\beta}) \bmod p$, we use

$$h(\boldsymbol{\beta}) = \frac{a([\gamma, \boldsymbol{\beta})])}{\prod_{\rho=1}^{r} f_\rho([\gamma, \boldsymbol{\beta}])^{e_\rho}} \bmod p \tag{6.11}$$

for a random $\gamma \in \mathbb{Z}_p$.

Let $\mathcal{C}_n = a \in \mathbb{Z}[x_1, \cdots, x_n]$. We have $\mathcal{C}_n = \text{cont}(\mathcal{C}_n) \cdot \text{pp}(\mathcal{C}_n) = \text{cont}(a) \cdot \text{pp}(a)$ where $\text{cont}(a)$ and $\text{pp}(a)$ are the content and primitive part of a respectively. After computing the irreducible factors of $\text{pp}(a)$, we also want to compute the irreducible factors of $\text{cont}(a) = \text{cont}(\mathcal{C}_n)$. Let $\mathcal{C}_{n-1} = \text{cont}(\mathcal{C}_n) \in \mathbb{Z}[x_2, \cdots, x_n]$. $\mathcal{C}_{n-1}$ is a polynomial with one fewer variable (without the main variable x_1). We now construct another black box $\mathbf{C}_{n-1} : \mathbb{Z}^{n-1} \times \{p\} \to \mathbb{Z}_p$ for $\mathcal{C}_{n-1}$ (see the next paragraph for how to construct $\mathbf{C}_{n-1}$) and use Algorithm CMBBSHL to compute its irreducible factors for $\text{pp}(\mathcal{C}_{n-1})$. Now we let $\mathcal{C}_{n-2} = \text{cont}(\mathcal{C}_{n-1})$. Construct

another black box $\mathbf{C}_{n-2} : \mathbb{Z}^{n-2} \times \{p\} \to \mathbb{Z}_p$ for $\mathcal{C}_{n-2}$ and use Algorithm CMBBSHL to compute the irreducible factors of $\mathrm{pp}(\mathcal{C}_{n-2})$. Repeat this process until $\mathcal{C}_0 = \mathrm{cont}(\mathcal{C}_1) \in \mathbb{Z}$. At this stage, $\mathcal{C}_0$ is an integer and it can be computed by calling the black box $\mathbf{C}_0 : \phi \times \{p\} \to \mathbb{Z}_p$ with different primes and Chinese remaindering.

In order to build the black box $\mathbf{C}_{n-1} : \mathbb{Z}^{n-1} \times \{p\} \to \mathbb{Z}_p$ for $\mathcal{C}_{n-1}$, we first construct a black box $\mathbf{F}_n : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{F}_n(\boldsymbol{\alpha}, p) = \mathrm{pp}(a)(\boldsymbol{\alpha}) \bmod p$. Let $f_{1,n}, \cdots, f_{r,n}$ be the irreducible factors of $\mathrm{pp}(a)$ found by Algorithm CMBBSHL. We have $\mathrm{pp}(a)(\boldsymbol{\alpha}) = \prod_{\rho=1}^{r} f_{\rho,n}(\boldsymbol{\alpha})^{e_\rho} \bmod p$. Next, the black box $\mathbf{C}_{n-1} : \mathbb{Z}^{n-1} \times \{p\} \to \mathbb{Z}_p$ can be computed by a division mod p, i.e.

$$\mathbf{C}_{n-1}(\boldsymbol{\beta}, p) = \frac{\mathbf{C}_n([\gamma, \boldsymbol{\beta}], p)}{\mathbf{F}_n([\gamma, \boldsymbol{\beta}], p)} \bmod p$$

for a fixed $\gamma \in \mathbb{Z}_p$ chosen at random. If $\mathbf{F}_n([\gamma, \boldsymbol{\beta}], p) = 0$ then FAIL is returned.

If $\deg(\mathcal{C}_{n-1}, x_2) = 0$, this means $\mathrm{pp}(\mathcal{C}_{n-1}) = 1$. In this case, we do not need to call algorithm CMBBSHL to compute $\mathrm{pp}(\mathcal{C}_{n-1})$. The black box $\mathbf{C}_{n-1}$ is constructed as

$$\mathbf{C}_{n-1}(\boldsymbol{\beta}, p) = \mathbf{C}_n([\gamma, \boldsymbol{\beta}], p)$$

for a fixed $\gamma \in \mathbb{Z}_p$ chosen at random and we proceed to the next level of recursion.

In Maple, the programs to make the black boxes $\mathbf{F}_n$ and $\mathbf{C}_{n-1}$ are written as Maple procedures `MakeBF` and `MakeCont`.

The input to `MakeBF` contains an array `FA` of the factors $f_{\rho,n}$ with their multiplicities e_ρ. In my implementation, the array `FA` is indexed from 0 to r', where r' is the number of irreducible factors that need to be Hensel lifted. `FA[0]` stores $[x_1, e_0]$, which represents a single term $x_1^{e_0}$ if it exists, since $x_1^{e_0}$ does not need to be Hensel lifted. If there is no such single term, `FA[0]` stores a 0. For $1 \le i \le r'$, `FA[i]` stores $[f_{i,n}, e_i]$, and $f_{i,n}$ is computed from Hensel lifting. `MakeBF` outputs a Maple procedure which is the black box $\mathbf{F}_n$. Inside the procedure `Fn`, each factor is evaluated at $\boldsymbol{\alpha}$ mod p, then the product $\prod_{\rho=1}^{r} f_{\rho,n}(\boldsymbol{\alpha})^{e_\rho}$ mod p is computed.

```
MakeBF := proc( FA::Array, r::nonnegint, X::list ) local N := nops(X), Fn;
    Fn := proc( alpha::Array, p::prime )
        local tmult:=1,i,t,tmon,tpow,teval,ii;
        for i from 0 to r do t := FA[i];
            if i = 0 then if t = 0 then next i; fi; fi;
            tmon := t[1]; tpow := t[2];
            teval := Eval( t[1], [seq(X[ii]=alpha[ii],ii=1..N)] ) mod p;
            teval := teval^tpow mod p;
            tmult := tmult*teval mod p;
        od;
        tmult;
```

```
    end;
end:
BF := MakeBF( FA, r, X );
```

The Maple program `MakeCont` has input the black box $\mathbf{C}_n$ (which is $\mathbf{B}$ for the first recursive step), the black box $\mathbf{F}_n$, a fixed $\gamma \in \mathbb{Z}_p$, and a prime p. It outputs the black box $\mathbf{C}_{n-1}$.

```
MakeCont := proc( Cn::procedure, Fn::procedure, gamma::integer, p::prime )
    local `Cn-1`;
    `Cn-1`:= proc( alpha::Array, p::prime )
        local na := numelems(alpha), alphaNew, g, i;
        alphaNew := Array(1..na+1); alphaNew[1] := gamma;
        for i to na do alphaNew[i+1] := alpha[i]; od;
        g := Fn( alphaNew, p );
        if g = 0 then return FAIL; fi;
        Cn( alphaNew, p )/g mod p;
    end;
end;
alpha0 := rand(p)();
BC := MakeCont( B, BF, alpha0, p );
```

6.3 Benchmarks

Before presenting the complexity analysis, we first show some timing benchmarks. We made a hybrid Maple and C implementation for algorithm CMBBSHL. The complete Maple code is on the website `http://www.cecm.sfu.ca/~mmonagan/code/CMBBSHL/` and Appendix A is the script to run algorithm CMBBSHL. Similarly to the monic and square-free case, the following sub-steps in each Hensel lifting step are coded in C to speed up computations:

- Step 8: Probes to the black box $\mathbf{B}$ and bivariate dense interpolation;
- Step 14: Evaluations of the factors $\hat{f}_{\rho,j-1}$;
- Step 17: Non-monic bivariate Hensel lifts (Algorithm 14);
- Step 23: Vandermonde solves.

At step 8 of Algorithm 13, the matrix A is converted to a list of polynomials to be passed into a C program for evaluations (BB eval). After evaluating each polynomial entry, another C program is called to calculate its determinant in $\mathbb{Z}_p$ (BB det). To compute $A_k = a_j(x_1, Y_k, x_j)$, we need $O(d_1 d_j)$ such evaluations and then perform a bivariate dense interpolation to get the bivariate image A_k.

At step 14, the polynomials $\hat{f}_{\rho,j-1}$ are evaluated by using two arrays for each factor. One array stores the coefficients $c_{\rho,ik}$, which is defined in step 1, the other array stores the monomial evaluations. This enables us to make use of the previous evaluation points, since our evaluation points $(\beta_2^k, \cdots, \beta_{j-1}^k)$ in step 7 are powers of the β_i's and $M(\beta_i^k) = (M(\beta_i))^k$.

Step 17 uses the cubic bivariate Hensel lifting (BHL) algorithm developed by Monagan and Paluck [48] in 2022. One BHL costs $O(d_1 d_j^2 + d_1^2 d_j)$ arithmetic operations in $\mathbb{Z}_p$.

The Vandermonde solves in step 23 use the classical algorithm of Zippel [62]. It does $O(s_{\rho,i}^2)$ arithmetic operations in $\mathbb{Z}_p$.

We present three timing benchmarks. All timings were obtained on the CECM gaby server which has 2 Intel Xeon E5-2660 8 core CPUs with 64 GB RAM. We used $p = 2^{62} - 57$ and $\tilde{N} = 4001$. We only used 1 core.

The first benchmark presents timings to compute the determinants of matrices B_n, where each B_n consists of four factors. For example, B_4 is of size 8×8 and it has 4 variables:

$$B_4 = \begin{bmatrix} uvw & v & uvw+v+w & \cdots & uvw+v \\ v & uvw & uvw+2v & \cdots & uvw+v \\ w & v & uvw+v+w & \cdots & v+w \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w & v & uvw+v+w & \cdots & 2vwx+2ux+3v+4w \end{bmatrix}.$$

$$\begin{aligned} \det(B_4) = -(&-v^2w^2x^2 + uvwx^2 + vw^2x - uwx + v^2 - 2vw + w^2) \\ &(v^2w^2x^2 + uvwx^2 + vw^2x + uwx - v^2 - 2vw - w^2) \\ &(u^2v^2w^2 + u^2vwx + uv^2w + uvx - v^2 - 2vw - w^2) \\ &(u^2v^2w^2 - u^2vwx - uv^2w + uvx - v^2 + 2vw - w^2). \end{aligned}$$

The number of terms in $\det(B_4)$ is 120, and each factor has 7 terms. And the leading coefficients in each variable is non-monic, e.g.

$$\begin{aligned} \text{lcoeff}(B,u) &= -v^6w^6x^4 + v^4w^4x^6 + v^4w^6x^2 - v^2w^4x^4, \\ \text{lcoeff}(B,v) &= u^4w^8x^4 - 2u^4w^6x^2 - 3u^2w^6x^4 + u^4w^4 + 6u^2w^4x^2 + w^4x^4 - 3u^2w^2 \\ &\quad - 2w^2x^2 + 1. \end{aligned}$$

All the matrices we used for our benchmarks are available online:
`http://www.cecm.sfu.ca/~mmonagan/code/BBfactor/`

Table 6.1 shows the CPU timings (in seconds) for our new algorithm CMBBSHL, compared with Maple and Magma's current best determinant and factorization algorithms. We used Maple 2022 and Magma V2.25-5 to compute the determinants of B_n and factored them. Maple 2022 uses Monagan and Tuncer's algorithm MTSHL [47] for factoring mul-

Table 6.1: CPU timings (in seconds) for computing the factors of $\det(B_n)$.

n	5	6	7	8	9
$N = 2n$	10	12	14	16	18
$\#f_1, \#f_2$	12,7	32,32	56,30	167,167	153,294
$\#f_3, \#f_4$	12,7	32,32	56,30	167,167	253,294
$\#\det(B_n)$	701	5162	79740	1716810	7490224
CMBBSHL tot	0.323	0.999	3.320	17.542	34.150
probes tot	1944	6156	18936	84240	143775
Maple det	0.455	7.880	382.80	$>$ 64 gigs	N/A
Maple fac	0.109	0.326	1.270	42.15	139.80
Maple tot	0.564	8.206	384.07	-	-
Magma det	1.680	6.290	594.60	$>$ 3h	N/A
Magma fac	0.120	0.480	33.140	N/A	N/A
Magma tot	1.800	6.770	627.74	N/A	N/A

N/A: Not attempted.

Table 6.2: Breakdown of timings for H.L. x_n for computing $\det(B_n)$.

n	5	6	7	8	9
$N = 2n$	10	12	14	16	18
H.L. x_n total	0.110	0.332	0.801	5.931	10.756
probes x_n	648	2115	4620	25753	45465
s (H.L. x_n)	9	25	31	131	201
BB tot	0.031	0.138	0.437	3.397	7.650
BB eval	0.016	0.076	0.315	2.586	5.773
BB det	0.008	0.038	0.069	0.512	1.189
Interp2var	0.002	0.004	0.008	0.065	0.116
Eval $\hat{f}_{\rho,j-1}$	0.005	0.017	0.020	0.072	0.143
BHL	0.038	0.109	0.211	1.263	1.989
VSolve	0.003	0.002	0.004	0.017	0.026

tivariate polynomials. The timings for Maple det were obtained by using Gentleman and Johnson's algorithm [26].

In Table 6.1, n is the number of variables of $a = \det(B_n)$. The size of matrix B_n is of $N \times N$ with $N = 2n$. $\#f_i$ $(i = 1, 2, 3, 4)$ is the number of terms in each factor of a. $\#\det(B_n)$ is the number of terms of $\det(B_n)$ in expanded form. CMBBSHL tot is the total time for our algorithm, and probes tot is the total number of probes to the black box $\mathbf{B}$ for CMBBSHL. Maple det is the time for determinant computation in Maple. Maple fac is the time for Maple's factorization. Similarly for the last section of Magma's timings. Our algorithm outperformed both Maple and Magma at $n = 5$. At $n = 7$, CMBBSHL is more than 100 times faster than Maple and 190 times faster than Magma. At $n = 8$, Maple ran out of memory at computing $\det(B_8)$ and CMBBSHL only took 17.5 seconds.

Table 6.2 shows a breakdown of timings for each subroutine for Hensel lifting the last variable x_n. The number s is the number of bivariate images needed for the last Hensel lifting step (s is defined in step 5 of Algorithm 13). BB tot is the total time for probes to the black box $\mathbf{B}$ (at step 8). BB eval is the time for evaluating the polynomial entries of the matrix B_n. BB det is the time for computing the determinant in $\mathbb{Z}_p$. Eval $\hat{f}_{\rho,j-1}$ is the time for evaluating the factors $\hat{f}_{\rho,j-1}$ at step 14. BHL is the time for bivariate Hensel lifts at step 17. VSolve is the time for Vandermonde solves at step 23.

The second benchmark shows the CPU timings for computing the factors of the determinant of Vandermonde matrices. This benchmark has a non-trivial content. The content itself is factored recursively by our algorithm. Let V_n be an $n \times n$ Vandermonde matrix. For example,

$$V_4 = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{pmatrix}. \tag{6.12}$$

$\det(V_4) = (x_1 - x_2)(x_1 - x_3)(x_1 - x_4)(x_2 - x_3)(x_2 - x_4)(x_3 - x_4)$. If $\det(V_4)$ is expanded, it has 24 terms. In general, there are $\binom{n}{2}$ linear factors for $\det(V_n)$.

In Table 6.3, n is the number of variables (is also $\dim(V_n)$), r is the number of factors of $\det(V_n)$, and $\#\det(V_n)$ is the number of terms of $\det(V_n)$ in its expanded form. CMBBSHL tot is the total time for algorithm CMBBSHL, and probes tot is the total number of probes to the black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ for algorithm CMBBSHL. Maple det is the time for Maple to compute the determinant of V_n by Gentleman and Johnson's algorithm [26]. Maple minor is the time for Maple to compute $\det(V_n)$ by the command `Determinant(V_n,method=minor)`. Maple fac uses Monagan and Tuncer's algorithm MTSHL [47] for factoring multivariate polynomials. We see that at $n = 9$, algorithm CMBBSHL outperforms Maple, and at $n = 11$, algorithm CMBBSHL is 170 times faster than Maple. Maple ran out of memory for computing $\det(V_{12})$, but algorithm CMBBSHL took only 5.19 seconds.

Table 6.4 shows the CPU timings for computing the primitive factors of the content at each recursive step. Recall that from Section 6.2, we defined $\mathcal{C}_n = a = \text{cont}(\mathcal{C}_n) \cdot \text{pp}(\mathcal{C}_n) \in$

Table 6.3: CPU timings (in seconds) for computing the factors of $\det(V_n)$.

$n = N$	7	8	9	10	11	12	13
$r = \binom{n}{2}$	21	28	36	45	55	66	78
$\#\det(V_n)$	5040	40320	362880	3628800	39916800	O/M*	N/A
CMBBSHL tot	0.336	0.649	1.137	1.990	3.290	5.190	8.175
probes tot	1328	2256	3597	5467	7975	11263	15479
pp(a) fac only	0.097	0.130	0.175	0.262	0.331	0.401	0.513
Maple det	0.061	0.100	0.446	5.700	45.07	$>$ 64 gigs	N/A
Maple minor	0.009	0.036	0.297	5.391	35.518	$>$ 64 gigs	N/A
Maple fac	0.012	0.068	0.882	17.96	523.80	N/A	N/A
Maple tot	0.021	0.104	1.179	23.351	559.318	N/A	N/A

N/A: Not attempted. O/M*: Out of memory at expanding the factors in Maple.

Table 6.4: CPU timings (in seconds) for computing the factors of pp(C_i).

$n = N$	7	8	9	10	11	12	13
CMBBSHL tot	0.366	0.649	1.137	1.990	3.290	5.190	8.175
probes tot	1328	2256	3597	5467	7975	11263	15479
pp(a) fac	0.097	0.130	0.175	0.262	0.331	0.401	0.513
pp(C_i) fac	0.098	0.180	0.263	0.369	0.593	0.808	1.140
($i = n-1, \cdots, 0$)	0.097	0.137	0.246	0.426	0.654	0.924	1.348
	0.045	0.100	0.213	0.411	0.541	0.913	1.348
	0.021	0.065	0.127	0.237	0.474	0.746	1.155
	0.005	0.026	0.070	0.153	0.351	0.577	0.984
	0.000	0.005	0.030	0.083	0.190	0.370	0.707
	0.002	0.000	0.007	0.036	0.100	0.267	0.495
	-	0.001	0.000	0.008	0.042	0.115	0.267
	-	-	0.002	0.000	0.009	0.051	0.142
	-	-	-	0.002	0.001	0.011	0.056
	-	-	-	-	0.002	0.000	0.012
	-	-	-	-	-	0.003	0.000
	-	-	-	-	-	-	0.003

$\mathbb{Z}[x_1, \cdots, x_n]$. And $\mathcal{C}_{n-1} := \mathrm{cont}(\mathcal{C}_n) = \mathrm{cont}(\mathcal{C}_{n-1}) \cdot \mathrm{pp}(\mathcal{C}_{n-1}) \in \mathbb{Z}[x_2, \cdots, x_n]$. For example,

$$\mathcal{C}_4 = \det(V_4) = \underbrace{(x_3 - x_4)(x_2 - x_3)(x_2 - x_4)}_{\mathcal{C}_3 = \mathrm{cont}(\mathcal{C}_4)} \underbrace{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)}_{\mathrm{pp}(\mathcal{C}_4)},$$

$$\mathcal{C}_3 = \underbrace{(x_3 - x_4)}_{\mathcal{C}_2 = \mathrm{cont}(\mathcal{C}_3)} \underbrace{(x_2 - x_3)(x_2 - x_4)}_{\mathrm{pp}(\mathcal{C}_3)},$$

$$\mathcal{C}_2 = \underbrace{x_3 - x_4}_{\mathrm{pp}(\mathcal{C}_2)},$$

$$\mathcal{C}_1 = \mathrm{cont}(\mathcal{C}_2) = 1,$$

$$\mathcal{C}_0 = \mathrm{cont}(\mathcal{C}_1) = 1.$$

Table 6.5: CPU timings (in seconds) for computing the factors of $\det(V_n)$ for larger n.

$n = N$	15	20	25	30	35	40
$r = \binom{n}{2}$	105	190	300	435	595	780
CMBBSHL tot	18.625	109.996	440.17	1376.793	3560.706	9057.977
probes tot	27311	85622	207912	429752	793809	1350786
pp(a) fac	0.791	2.246	5.891	13.968	29.597	57.745
H.L. x_n	0.055	0.117	0.256	0.467	0.800	1.487
probes x_n	465	820	1275	1830	2485	3240
s (H.L. x_n)	1	1	1	1	1	1
BB tot	0.024	0.070	0.156	0.353	0.650	1.224
BB eval	0.015	0.039	0.090	0.208	0.368	0.709
BB det	0.009	0.031	0.066	0.145	0.282	0.515
Interp2var	0.001	0.002	0.004	0.009	0.015	0.027
Eval $\hat{f}_{\rho_j-1}$	0.002	0.002	0.003	0.004	0.004	0.005
BHL	0.004	0.005	0.008	0.009	0.010	0.011
VSolve	0.004	0.003	0.006	0.009	0.007	0.012

At each recursive call, the factors of pp($\mathcal{C}_i$) are computed. In Table 6.4, pp(a) fac is the timing for computing the primitive factors of $\det(V_n)$. pp(C_i) fac presents the timings for algorithm CMBBSHL to compute the factors of pp(C_i) for $i = n-1, \cdots, 0$.

Our software has the capability to compute the factors of pp(a) only. Here $\text{pp}(a) = \prod_{j=2}^{n}(x_1 - x_j)$ is much smaller than the content $\text{cont}(a) = \prod_{2\le i<j\le n}(x_i - x_j)$. In Table 6.4, for $n = 13$, it took only 0.513 seconds to factor pp(a) and $8.175 - 0.153 = 7.662$ seconds to be able to factor cont(a).

Table 6.5 shows more timings for $\det(V_n)$ with larger n. pp(a) fac is the time for computing the factors of the primitive part of $a = \det(V_n)$. H.L. x_n is the total time for Hensel lifting the last variable x_n. The last block of rows shows breakdown of timings for each subroutine at Hensel lifting x_n.

In the first benchmark, $\det(B_n)$ is non-monic but square-free and primitive. In the second benchmark, $\det(V_n)$ is non-monic and non-primitive but still square-free. The matrices in the first two benchmarks are also relatively small. Our third benchmark is for large matrices and their determinants are non-monic, non-square-free and non-primitive. The matrices come from Dixon matrices which come from solving polynomial systems of equations. Table 6.6 presents timings for 7 different matrices with various n and N. For example, heron3d is 13×13 and it has the following determinant with 7 variables:

$$\det(A) = 64as^7(as - bs + cs)(as - bs - cs)(as + bs + cs)(as+bs-cs)$$
$$\underbrace{(as^4es^2 + as^2bs^2cs^2 - \cdots - cs^2es^2fs^2 + 144vo^2)^2}_{23 \text{ terms}}.$$

In Table 6.6, n is the number of variables, the size of the matrices is $N \times N$. r is the number of square-free factors of a. $d_j = \deg(a, x_j)$ are the degrees of a in each variable x_j, computed prior to Hensel lifting. $\#f_i$ are the number of terms of each square-free factor and e_i are the corresponding powers of the factors. The number $\max \lambda_\rho$ is the maximum of λ_ρ $(1 \leq \rho \leq r)$, computed from rational number reconstruction. CMBBSHL tot is the total time for algorithm CMBBSHL including computing the factors of the content. pp(a) fac is the time for computing the primitive factors only. We compared our timings with Maple's determinant computation and factorization. Using Gentleman and Johnson's algorithm [26] to compute the determinant, Maple ran out of memory for computing the determinant of `heron4d`, and algorithm CMBBSHL took only 43.809 seconds. The timings in Table 6.6 were all computed with the variables sorted lexicographically. With different variable orderings, timings can be quite different. For example, for `heron4d`, as also presented in Chapter 2 using $X = [es, bs, cs, ds, fs, as, gs, hs, is, js, vo]$, it only took 17.56 seconds in total.

Table 6.7 shows a breakdown of timings for our algorithm CMBBSHL at the last Hensel lifting step. BB tot includes the timings of BB eval (evaluations of the polynomial entries) and BB det (determinant computation in $\mathbb{Z}_p$). The matrix robotarms (b_2) has a larger number of terms (about 100) in each matrix entry. We can see that BB tot is much larger for robotarms (b_2) than heron4d. heron5d is a much larger matrix, and BB tot is the bottleneck.

Table 6.6: Timings (in seconds) for computing the determinant of Dixon matrices.

	heron3d	heron4d	robotarms (b_1)	robotarms (b_2)	robotarms (t_1)	robotarms (t_2)	heron5d
n	7	11	8	8	8	8	16
$N \times N$	13×13	63×63	16×16	20×20	16×16	12×12	399×399
$d_j = \deg(a, x_j)$ $(j = 1, \cdots, n)$	19,12,12, 8,8,8,4	89,26,26, 12,12,12 8,8,8,8,8	16,64,128,44 36,16,16,16	20,80,80,36, 40,20,12,12	16,64,32,56, 32,128,16,16	12,48,26,24 16,48,8,8	2159,328,328,144, 144,144,64,64, 64,64,32,32, 32,32,32,16
r	6	4	8	10	7	9	8
$\#f_i$ $(i = 1, \cdots, r)$	3,23,3, 3,1,3	22,1, 6,131	1,1,2,39, 2,1,4,7	2,1,1,2,6, 2,2124,4,7,1	2,1,30,6, 2,7,7	2,2,1,1,1 6,2,2316,7	823,130,22,3 3,3,3,1
e_i $(i = 1, \cdots, r)$	1,2,1, 1,7,1	2,37, 7,4	8,24,48,8 24,4,4,4	3,8,28,30,7, 31,1,4,4,4	48,16,8,8, 16,4,4	1,18,4,8,12 3,19,1,4	8,8,20,46 46,46,1831
$\#\det(A)$	525	37666243	O/M*	88687776	O/M*	16963876	N/A
$\max \lambda_\rho$	1	1	1	169	2	4	1
CMBBSHL tot	0.685	43.809	169.851	935.219	350.809	325.647	165208.747
probes tot	5701	201183	99652	468627	131250	325500	36008392
pp(a) fac	0.683	43.804	18.972	854.153	43.178	302.374	165106.278
probes pp(a)	5699	201181	11448	438808	16626	312169	-
Maple det	0.614	O/M	N/A	N/A	N/A	N/A	N/A
Maple minor	0.006	0.383	O/M	O/M	O/M	O/M	N/A
Maple fac	0.084	O/M	N/A	N/A	N/A	N/A	N/A
Maple tot	0.620	-	-	-	-	-	-

N/A: Not attempted. O/M: Out of memory. O/M*: Out of memory when expanding the factors in Maple.

Table 6.7: Breakdown of timings (in seconds) for computing the determinant of Dixon Matrices.

	heron3d	heron4d	robotarms (b_1)	robotarms (b_2)	robotarms (t_1)	robotarms (t_2)	heron5d
n	7	11	8	8	8	8	16
$N \times N$	13×13	63×63	16×16	20×20	16×16	12×12	399×399
H.L. x_n tot	0.146	10.431	1.451	360.580	2.958	111.605	14347.878
probes x_n	930	41112	901	178368	1173	114088	-
s	13	85	3	806	3	974	571
BB tot	0.039	9.303	1.398	349.916	2.910	103.615	13771.116
BB eval	0.022	4.764	1.369	339.965	2.888	100.207	7254.237
BB det	0.008	3.720	0.029	6.096	0.022	1.716	6414.483
Interp2var	0.000	0.061	0.003	0.412	0.003	0.153	17.574
Eval $\hat{f}_{\rho,j-1}$	0.029	0.029	0.000	0.418	0.000	0.295	0.576
BHL	0.029	0.160	0.000	2.357	0.000	1.758	450.033
VSolve	0.001	0.002	0.000	0.366	0.002	0.412	0.031

6.4 Complexity analysis with failure probabilities

Algorithm CMBBSHL can return FAIL with a low probability. It could also return an incorrect answer (not a FAIL) with a low probability. If p divides any integer coefficient of any irreducible factor $f_\rho \in \mathbb{Z}[x_1, \cdots, x_n]$, algorithm CMBBSHL returns an incorrect answer. The failure probability bound is pessimistic, and in practice, I have never observed a failure.

Example 17. Let $p = 2^{62} - 57$. Let $a = f_1 f_2 \in \mathbb{Z}[x, y, z]$, where $f_1 = 2xy + 5$ and $f_2 = 3x^2z + pxy + 7$. Suppose $\boldsymbol{\alpha} = (9, 5)$. Factoring $a(x, \boldsymbol{\alpha})$ gives $(18x + 5)(15x^2 + 9px + 7)$. Thus, $\hat{f}_{1,1} = 18x + 5$ and $\hat{f}_{2,1} = 15x^2 + 7$. After the first BHL (the second Hensel lifting step), algorithm CMBBSHL outputs

$$\hat{f}_{1,2} = 2xy + 5 \text{ and } \hat{f}_{2,2} = 15x^2 + 7.$$

After the third Hensel lifting step,

$$\hat{f}_{1,3} = 2xy + 5 \text{ and } \hat{f}_{2,3} = 3x^2z + 7.$$

After rational number reconstruction, we get the factors $f_\rho = \hat{f}_{\rho,3}$ for $\rho = 1, 2$. Because the Hensel lifting is done mod p, the second term pxy in f_2 is missing.

Proposition 6.4.2 gives a bound for the probability of CMBBSHL returning FAIL at the j^{th} Hensel lifting step (Algorithm 13). Proposition 6.4.3 gives a bound for the probability of CMBBSHL returning an incorrect answer due to the fact that p divides at least one integer coefficient of any irreducible factor.

In order to prove Proposition 6.4.2, we need Lemma 6.4.1 (Proposition 1(ii) in Section 6 of Chapter 3 in [14]) on the theory of resultants.

Lemma 6.4.1. *Let $\mathbb{F}$ be a field. Let $A, B \in \mathbb{F}[x_1, \cdots, x_n]$ such that $\deg(A, x_1) > 0$ and $\deg(B, x_1) > 0$. Then*

$$\operatorname{res}(A, B, x_1) = 0 \iff \deg(\gcd(A, B), x_1) > 0. \tag{6.13}$$

Proposition 6.4.2. *Let p be a large prime. Let r be the number of factors of $\operatorname{sqf}(a)$. Let $d = \deg(a)$, $\tilde{d} = \deg(\operatorname{sqf}(a))$, $\tilde{d}_j = \deg(\operatorname{sqf}(a), x_j)$ and s be the number defined at step 5 in Algorithm 13. Let $\#\hat{f}_{\rho,j-1}$ denote the number of terms in the input factors $\hat{f}_{\rho,j-1}$ at the j^{th} Hensel lifting step of Algorithm 13. Then, CMBBSHL (Algorithm 13) fails to compute*

$\hat{f}_{\rho,j} \in \mathbb{Z}_p[x_1, \cdots, x_j]$ *with a probability less or equal than*

$$\underbrace{\frac{((r^2-r+2)\tilde{d}^2+2d+\tilde{d})s^2+\left(2\tilde{d}^2+\tilde{d}\left(\sum_{\rho=1}^{r}\#\hat{f}_{\rho,j-1}+1\right)+2d\right)s}{2(p-1)}}_{\textit{steps } 4,9,11,15,16}+\underbrace{\frac{\tilde{d}_j^2\sum_{\rho=1}^{r}\#\hat{f}_{\rho,j-1}}{p-\tilde{d}_j+1}}_{\textit{Lemma } 3.3.2}. \tag{6.14}$$

Proof. For step 11, let $\bar{A}_k = A_k/g_k \in \mathbb{Z}_p[x_1, x_j]$ and let $\overline{\frac{\partial A_k}{\partial x_1}} = \frac{\partial A_k}{\partial x_1}/g_k \in \mathbb{Z}_p[x_1, x_j]$, where $g_k = \gcd(A_k, \frac{\partial A_k}{\partial x_1}) \in \mathbb{Z}_p[x_1, x_j]$ from step 10. Then, by Lemma 6.4.1,

$$\deg\left(\gcd\left(\bar{A}_k, \overline{\frac{\partial A_k}{\partial x_1}}\right), x_1\right) > 0 \Leftrightarrow \operatorname{res}\left(\bar{A}_k, \overline{\frac{\partial A_k}{\partial x_1}}, x_1\right) = 0.$$

Let $g_j = \gcd(a_j, \frac{\partial a_j}{\partial x_1}) \in \mathbb{Z}_p[x_1, \cdots, x_j]$. Let $\bar{a}_j = a_j/g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$. Let $\overline{\frac{\partial a_j}{\partial x_1}} = \frac{\partial a_j}{\partial x_1}/g_j \in \mathbb{Z}_p[x_1, \cdots, x_j]$. Define $R = \operatorname{res}\left(\bar{a}_j, \overline{\frac{\partial a_j}{\partial x_1}}, x_1\right) \in \mathbb{Z}_p[x_2, \cdots, x_j]$. Let $R_k = R(x_2^k, \cdots, x_{j-1}^k, x_j)$ and let $S = \prod_{k=1}^{s} R_k$.

Algorithm CMBBSHL step j fails at step 11 if $R(Y_k, x_j) = R_k(\beta_2, \cdots, \beta_{j-1}, x_j) = 0$ for some k. Let $(\beta_2, \cdots, \beta_j)$ be chosen at random from $(\mathbb{Z}_p \backslash \{0\})^{j-1}$,

$$\Pr[R(Y_k, x_j) = 0 \text{ for some } k] = \Pr[S(\beta_2, \cdots, \beta_{j-1}, x_j) = 0].$$

Let $d_{S_j} = \deg(S, x_j)$ and $S = \sum_{i=0}^{d_{S_j}} c_i(x_2, \cdots, x_{j-1})x_j^i$, where $c_i \in \mathbb{Z}_p[x_2, \cdots, x_{j-1}]$. Then,

$$\begin{aligned}
&\Pr[S(\beta_2, \cdots, \beta_{j-1}, x_j) = 0] \\
=&\Pr\left[c_0(\beta_2, \cdots, \beta_{j-1}) = 0 \wedge c_1(\beta_2, \cdots, \beta_{j-1}) = 0 \wedge \cdots \wedge c_{d_{S_j}}(\beta_2, \cdots, \beta_{j-1}) = 0\right] \\
\leq&\Pr[c_k(\beta_2, \cdots, \beta_{j-1}) = 0] \text{ for some } k \in \{0, 1, \cdots, d_{S_j}\} \text{ with } c_k \neq 0 \\
\leq&\frac{\deg(c_k)}{p-1} \text{ by Lemma 1.6.4} \\
\leq&\frac{\deg(S)}{p-1}.
\end{aligned}$$

Now,

$$\deg(S) = \sum_{k=1}^{s} \deg(R_k) \leq \sum_{k=1}^{s} 2k\tilde{d}^2 = \tilde{d}^2 s(s+1).$$

Thus, CMBBSHL (Algorithm 13) step j fails at step 11 with a probability less than

$$\frac{\tilde{d}^2 s(s+1)}{p-1}.$$

For step 9, let $L_1 := \mathrm{LC}(a_j, x_1) \in \mathbb{Z}_p[x_2, \cdots, x_j]$ and $L_j := \mathrm{LC}(a_j, x_j) \in \mathbb{Z}_p[x_1, \cdots, x_{j-1}]$. Algorithm CMBBSHL fails at step 9 if $L_1(Y_k, x_j) = 0$ or $L_j(Y_k, x_1) = 0$ for some k.

Let $L_{1,k} := L_1(x_2^k, \cdots, x_{j-1}^k, x_j)$ and $S_1 = \prod_{k=1}^{s} L_{1,k}$. By the same argument for step 11 and Lemma 1.6.4,

$$\Pr[L_1(Y_k, x_j) = 0 \text{ for some } k] = \Pr[S_1(\beta_2, \cdots, \beta_{j-1}, x_j) = 0] \leq \frac{\deg(S_1)}{p-1}.$$

We have $\deg(L_1) \leq d - d_1$, hence $\deg(L_{1,k}) \leq kd - d_1$. Thus,

$$\deg(S_1) = \sum_{k=1}^{s} \deg(L_{1,k}) \leq \sum_{k=1}^{s} (kd - d_1) = \frac{ds(s+1)}{2} - sd_1 < \frac{ds(s+1)}{2}.$$

Similarly for $\mathrm{LC}(a_j, x_j)$. Thus,

$$\Pr[\text{step 9 fails at step j}] < \frac{ds(s+1)}{p-1}.$$

The proof for step 15 is similar to step 9 above. The proofs for failure probabilities at steps 4 and 16 follow from [46] and Section 4.4 of this Thesis. And we have the following:

$$\begin{aligned}
\Pr[\text{step 4 fails at step j}] &< \frac{\tilde{d}s \sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1}}{2(p-1)}, \\
\Pr[\text{step 15 fails at step j}] &< \frac{\tilde{d}s(s+1)}{2(p-1)}, \\
\Pr[\text{step 16 fails at step j}] &< \frac{\tilde{d}^2 s^2 r(r-1)}{2(p-1)}.
\end{aligned}$$

Adding up, we get the first term in (6.14). The second term in (6.14) comes from the weak SHL assumption (Lemma 3.3.2). □

Proposition 6.4.3. *Let p be a 63-bit prime, i.e. $p \in (2^{62}, 2^{63})$. Let $\mathbb{P}_{63} = \{$all 63-bit primes$\}$. Let $f_\rho = \sum_{i=1}^{\#f_\rho} c_{\rho,i} \cdot x_1^{e_{i_1}} \cdots x_n^{e_{i_n}}$ for $1 \leq \rho \leq r$, where $c_{\rho,i} \neq 0$, $c_{\rho,i} \in \mathbb{Z}$, and $(e_{i_1}, \cdots, e_{i_n}) \in \mathbb{N}^n$. Let $\chi_\rho = \{i \in \mathbb{Z} \mid |c_{\rho,i}| \geq p\}$ and let $\#f_{\rho,p} = |\chi_\rho|$ for $1 \leq \rho \leq r$. Let $h_\rho = \|f_\rho\|_\infty$ for $1 \leq \rho \leq r$. Let $h_{\max} = \max_{\rho=1}^{r} h_\rho$. Then,*

$$\Pr[p \mid \textit{at least one } c_{\rho,i} \textit{ in any } f_\rho] \leq \frac{1}{|\mathbb{P}_{63}|} \left\lfloor \frac{\log_2(h_{\max})}{62} \right\rfloor \sum_{\rho=1}^{r} \#f_{\rho,p}. \tag{6.15}$$

Proof.

$$\begin{aligned}\Pr[p \mid \text{at least one } c_{\rho,i} \text{ in } f_\rho, i \in \chi_\rho] &\le \sum_{i\in\chi_\rho} \Pr[p|c_{\rho,i}] \\ &\le \sum_{i\in\chi_\rho} \left\lfloor \frac{\log_2(|c_{\rho,i}|)}{62} \right\rfloor \frac{1}{|\mathbb{P}_{63}|} \\ &\le \left\lfloor \frac{\log_2(h_\rho)}{62} \right\rfloor \frac{\#f_{\rho,p}}{|\mathbb{P}_{63}|}.\end{aligned}$$

Thus

$$\begin{aligned}\Pr[p \mid \text{at least one } c_{\rho,i} \text{ in any } f_\rho] &\le \sum_{\rho=1}^{r} \left\lfloor \frac{\log_2(h_\rho)}{62} \right\rfloor \frac{\#f_{\rho,p}}{|\mathbb{P}_{63}|} \\ &\le \frac{1}{|\mathbb{P}_{63}|} \left\lfloor \frac{\log_2(h_{\max})}{62} \right\rfloor \sum_{\rho=1}^{r} \#f_{\rho,p}.\end{aligned}$$

□

For example, consider the matrix $BL9$ in Table 6.8 (benchmark for the large integer case). We have $\log_2(h_{\max}) = 150.907$, $\#f_{1,p} \le 153$, and $\#f_{2,p} \le 294$. $|\mathbb{P}_{63}| \approx 1.039 \times 10^{17}$. Thus

$$\frac{1}{|\mathbb{P}_{63}|} \left\lfloor \frac{\log_2(h_{\max})}{62} \right\rfloor \sum_{\rho=1}^{r} \#f_{\rho,p} \approx 8.604 \times 10^{-15}.$$

Theorem 6.4.4 gives the complexity of CMBBSHL (Algorithm 13).

Theorem 6.4.4. *Let p be a large prime and $\tilde{N} < p$, $\tilde{N} \in \mathbb{Z}^+$. Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ and $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n) \in \mathbb{Z}_p^{n-1}$ be randomly chosen such that $0 < \alpha_i < \tilde{N}$. Suppose $\boldsymbol{\alpha}$ is Hilbertian and condition (i) of the input of CMBBSHL is satisfied. Then, if algorithm CMBBSHL returns an answer that is not FAIL, the total number of arithmetic operations in $\mathbb{Z}_p$ in the worst case for lifting $\hat{f}_{\rho,1}$ to $\hat{f}_{\rho,n}$ using Algorithm 13 $n-1$ times is*

$$O\left((n-2)s_{\max}d_{\max}\left(\sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1} + d_1^2 + d_1 d_{\max} + d_1\, C(\text{probe } \boldsymbol{B})\right)\right). \tag{6.16}$$

where $d_1 = \deg(a, x_1)$, $d_{\max} = \max_{j=2}^{n}(\deg(a, x_j))$, $s_{\max}$ is defined in Definition 5.3.1 and $C(\text{probe } \boldsymbol{B})$ is the number of arithmetic operations in $\mathbb{Z}_p$ for one probe to the black box $\boldsymbol{B}$. The total number of probes to the black box is $O(nd_1d_{\max}s_{\max})$.

Proof. Let $d_j = \deg(a, x_j)$, $\tilde{d}_1 = \deg(\text{sqf}(a), x_1)$ and $\tilde{d}_j = \deg(\text{sqf}(a), x_j)$. For step 8, we use dense interpolation to get a bivariate image $a_j(x_1, Y_k, x_j)$. Thus, it requires $O(d_1 d_j)$ probes to $\mathbf{B}$ and $O(d_1 d_j^2 + d_1^2 d_j)$ arithmetic operations in $\mathbb{Z}_p$ for one image. The total cost for step 8 for CMBBSHL step j is $O(s(d_1 d_j \text{C(probe } \mathbf{B})))$ plus $O(s(d_1^2 d_j + d_1 d_j^2))$ operations in $\mathbb{Z}_p$ for all dense interpolations.

For step 10, we can use Brown's GCD algorithm [6] for GCDs in $\mathbb{Z}_p[x_1, x_j]$ which costs $O(d_1^2 d_j + d_1 d_j^2)$ arithmetic operations in $\mathbb{Z}_p$. An alternative with the same asymptotic cost would be to use the bivariate Hensel lifting of Monagan and Paluck [48]. The total cost for step 10 for step j of Algorithm CMBBSHL is $O(s(d_1^2 d_j + d_1 d_j^2))$ operations in $\mathbb{Z}_p$.

The proofs of the following steps follow from Section 4.4 of this Thesis. We give the total count of arithmetic operations in $\mathbb{Z}_p$ for step j:

Step 14 costs $O(s \sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1})$.

Step 17 costs $O(s(\tilde{d}_1^2 \tilde{d}_j + \tilde{d}_1 \tilde{d}_j^2)) \subseteq O(s(d_1^2 d_j + d_1 d_j^2))$.

Step 23 costs $O(s\tilde{d}_j \sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1})$.

Adding up, we get the total number of arithmetic operations in $\mathbb{Z}_p$ for step j of algorithm CMBBSHL:

$$O\left(s\left(d_1^2 d_j + d_1 d_j^2\right) + s\tilde{d}_j \sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1} + sd_1 d_j \mathrm{C}(\text{probe } \mathbf{B})\right). \tag{6.17}$$

And (6.16) follows from (6.17).

□

6.5 Large integer coefficients

For large integer coefficients, we need to use a larger prime than a 63-bit prime. In this case, the C codes can no longer be used since they are coded using signed 63-bit integers. However, algorithm CMBBSHL has an option to run every subroutine in Maple instead. To enable this, set the option `MapleCode := [Maple,Maple,Maple,Maple]`, so each of the 4 major sub-steps is computed in Maple.

We created some test examples to compute $\det(BL_n)$ for $n = 4, \cdots, 9$. The matrix BL_n is created similarly to the matrix B_n in Section 6.3 except the coefficients of $\det(BL_n)$ are large (greater than 2^{64}). The matrices BL_n for $n = 4, \cdots, 9$ are available on the website `http://www.cecm.sfu.ca/~mmonagan/code/CMBBSHL/`.

For example, BL_5 is a 10×10 matrix and it looks like:

$$\begin{pmatrix} 984961191x_1 & 303065689x_2 & 236721026x_3 & \cdots \\ 303065689x_2 & 984961191x_1 & 303065689x_2 & \cdots \\ 303065689x_2 + 236721026x_3 & 984961191x_1 + 303065689x_2 & 984961191x_1 + 303065689x_2 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}.$$

The max-norm for $\det(BL_5)$ in its expanded form is $2^{298.755}$ (see Table 6.8).

Table 6.8 shows timings for computing the factors of $\det(BL_n)$ for $n = 5, \cdots, 9$. In Table 6.8, n is the number of variables, N is the number of rows (or columns) of BL_n, $\#f_i$ denote the number of terms in each square-free factor, and e_i are the corresponding powers. For each $\det(BL_n)$, there are two irreducible factors and each factor is squared. $\#\det(BL_n)$

is the number of terms of $\det(BL_n)$ if expanded. Since the max-norm of $\det(BL_n)$ and p are quite large, we present them in log base 2. As n increases, we used larger primes. If algorithm CMBBSHL returns a FAIL, the size of the prime is then doubled. Our algorithm outperforms Maple at $n = 6$. At $n = 8$, our algorithm is over 290 times faster than Maple's determinant and factorization. Table 6.9 shows the breakdown of timings at Hensel lifting the last variable x_n. The bottleneck is probes to the black box $\mathbf{B}$.

Table 6.8: Timings (in seconds) for computing $\det(BL_n)$

n	5	6	7	8	9
$N = 2n$	10	12	14	16	18
$\#f_i$	12,7	32,32	56,30	167,167	153,294
e_i	2,2	2,2	2,2	2,2	2,2
$\#\det(BL_n)$	297	1873	11463	73184	455236
$\log_2(\#f_i)$	84.768	89.067	118.916	119.664	150.907
$\log_2(\|\det(BL_n)\|_\infty)$	298.755	360.788	421.960	480.880	548.934
$\log_2(p)$	298.755	360.222	420.912	479.133	546.178
CMBBSHL tot	2.762	17.263	67.526	355.197	1083.36
content tot	0.011	0.006	0.009	0.014	0.023
probes tot	1487	5527	13757	47857	109689
Maple det	0.842	18.69	858	104256	$>$ 48hrs
Maple fac	0.118	0.263	5.77	88.63	N/A
Maple tot	0.960	18.953	863.77	104344.63	N/A

Table 6.9: Breakdown of timings for H.L. x_n for $\det(BL_n)$.

n	5	6	7	8	9
$N = 2n$	10	12	14	16	18
H.L. x_n total	0.632	3.942	9.565	59.609	123.810
s (H.L. x_n)	5	18	24	93	130
BB eval	0.179	1.016	2.459	14.392	28.057
BB det	0.404	2.706	6.767	43.428	92.870
Eval $\hat{f}_{\rho_j-1}$	0.000	0.001	0.001	0.055	0.188
BHL	0.009	0.035	0.056	0.259	0.375
VSolve	0.002	0.012	0.018	0.177	0.238

Chapter 7

Implementation details

In this chapter, I present some implementation details for algorithm CMBBSHL. We used a hybrid of Maple and C codes. The main program is coded in Maple. Some sub-programs are coded in C. All four major subroutines in the j^{th} Hensel lifting step of CMBBSHL have been coded in C.

Algorithm CMBBSHL is a black box factorization algorithm. Thus, we give particular details on black box construction and the step (the only step) that CMBBSHL calls the black box, i.e., the bivariate dense interpolation in step 8 of Algorithm 13. Section 7.1 gives a demonstration of how a Maple + C hybrid implementation works. Section 7.2 presents details on black box construction. In particular, given a matrix A with multivariate polynomial entries, we construct a black box that computes the determinant of A. Section 7.3 presents Bareiss' $\mathcal{O}(n^2)$ algorithm for computing the determinant of a Toeplitz matrix. Section 7.4 shows implementation details for bivariate dense interpolation.

For the implementations described in this chapter, I am the author of the codes. For the implementation of bivariate dense interpolation in C, which I discussed with Prof. Monagan, he gave me an idea of how to store the matrices efficiently.

7.1 Maple + C implementation

Within a Maple program, an external C program can be called by using the command `define_external`, which produces a Maple procedure that links to the external C program.

For example, the C program `det64s` computes the determinant of a matrix A mod a prime p. The input variables to the C program are a one-dimensional C array of size n^2 representing a matrix A stored in *row-major* order, an integer $n < 2^{31}$ which has data type `int`, and an integer $p < 2^{63}$ which has data type `long long int`.

The following Maple code is used to create an interface to the external C function `det64s`. The input variables of `Det64s` are a square matrix A with entries of data type `long long int`, the number of rows (and columns) of the matrix, n, which has data type `int`, and a prime p, which has data type `long long int`.

```
> Det64s := define_external( 'det64s',
>   AA::ARRAY(1..nn,1..nn,integer[8],order=C_order),
>   nn::integer[4],
>   pp::integer[8],
>   LIB=cat(CCodeDir,"detinterp4.so"),
>   RETURN::integer[8]):
```

The option `order=C_order` means that the *row-major* order is used to store the matrix A as a one-dimensional array. This means that for an $n \times n$ matrix A, the entries are stored as

$$[A_{1,1}, A_{1,2}, \cdots, A_{1,n}, A_{2,1}, \cdots, A_{2,n}, \cdots, A_{n,1}, \cdots, A_{n,n}].$$

After creating a Maple procedure `Det64s`, it can be used as any other Maple procedure. `Det64s` computes the determinant of a matrix mod a prime p. It calls the external C program `det64s` to do it. Running a test example gives the following output:

```
> CCodeDir := "./":
> n := 3: p := 101:
> A := Matrix([[1,2,3],[1,4,9],[1,8,27]],datatype=integer[8],order=C_order);
                                    [1    2     3]
                                    [             ]
                               A := [1    4     9]
                                    [             ]
                                    [1    8    27]

> d := Det64s(A,n,p); # Calls the external C program det64s
                                     d := 12
> A; # A has been modified after Gaussian elimination
                                   [1    2    3]
                                   [           ]
                                   [0    1    3]
                                   [           ]
                                   [0    0    1]

# Re-define the matrix A and check in Maple:
> A := Matrix([[1,2,3],[1,4,9],[1,8,27]]):
> Det(A) mod p;
                                        12
```

To compile the C code, the following commands are used in LINUX:

```
gcc -c -O3 -fpic detinterp4.c
gcc -shared -o detinterp4.so detinterp4.o
```

Now `detinterp4.c` which contains the C program `det64s` has been compiled. The output file `detinterp4.so` must be saved in the directory `CCodeDir`.

7.2 Black box construction

Given a matrix A with multivariate polynomial entries $a_{ij} \in \mathbb{Z}[x_1, \cdots, x_n]$, we want to construct a modular black box $\mathbf{B} : \mathbb{Z}^n \times \{p\} \to \mathbb{Z}_p$ s.t. $\mathbf{B}(\boldsymbol{\alpha}, p) = \det(A) \bmod p$.

Calling the black box $\mathbf{B}$ involves two steps: BB eval and BB det. BB eval evaluates the entries a_{ij} at $\boldsymbol{\alpha}$ mod p. The output of BB eval is stored in a matrix $\tilde{A}$ with $\tilde{a}_{ij} \in \mathbb{Z}_p$. Then, BB det computes $\det(\tilde{A}) \bmod p$. Both BB eval and BB det have been coded in C. My supervisor Prof. Monagan contributed the C code for BB eval, as well as BB det for the general case, i.e. the C program `det64s` which computes $\det(\tilde{A}) \bmod p$ by Gaussian elimination. For the special case of symmetric Toeplitz matrices, I coded the C program for Bareiss' $\mathcal{O}(n^2)$ algorithm [1] for determinant computation, described in the next section.

In order to construct the black box $\mathbf{B}$, I wrote a Maple procedure `MakeBBdet_C`. It outputs an anonymous Maple procedure which is the black box $\mathbf{B}$. The anonymous procedure calls two external C programs (`EVALMOD1` is for BB eval and `det64s` is for BB det).

```
MakeBBdet_C := proc( A::Matrix, VarPerm::list )
    local n,X,N,Xnew,i,AL,AA;
    n := LinearAlgebra:-RowDimension(A);
    X := convert(indets(A),list);
    N := nops(X);
    Xnew := [seq(X[VarPerm[i]],i=1..N)];
    AL := convert(A,list);
    AA := Array(1..n,1..n,datatype=integer[8],order=C_order);
    proc( alpha::Array, p::prime ) global CNT;
        CNT++; # CNT = no. of probes to B
        EVALMOD1(AL, Xnew, alpha, AA, p); # Evaluate A at alpha mod p in C
        Det64s(AA, n, p); # Compute det(AA) mod p in C
    end;
end:
```

The second argument of `MakeBBdet_C`, `VarPerm`, is an input of an arbitrary variable permutation of choice. This is because algorithm CMBBSHL allows the user to choose any variable ordering as an input.

7.3 Bareiss' $\mathcal{O}(n^2)$ algorithm for computing the determinant of a symmetric Toeplitz matrix

Bareiss' algorithm [1] for computing the determinant of a Toeplitz matrix costs only $\mathcal{O}(n^2)$ arithmetic operations, whereas Gaussian elimination costs $\mathcal{O}(n^3)$.

Bareiss' algorithm transforms a Toeplitz matrix (not necessarily symmetric) A into a lower triangular matrix $A^{(-n)}$ and an upper triangular matrix $A^{(n)}$ by successive steps $A^{(0)}, A^{(\pm 1)}, A^{(\pm 2)} \cdots, A^{(\pm n)}$. Suppose a Toeplitz matrix A has the form

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & \cdots & a_n \\ a_{-1} & a_0 & a_1 & \cdots & a_{n-1} \\ a_{-2} & a_{-1} & a_0 & \cdots & a_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{-n} & a_{-(n-1)} & a_{-(n-2)} & \cdots & a_0 \end{pmatrix}. \tag{7.1}$$

Then, at the i$^{\text{th}}$ step, $A^{(-i)}$ and $A^{(i)}$ take the form

$$A^{(-i)} = \begin{pmatrix} a_0^{(0)} & a_1^{(0)} & \cdots & & & & & & a_n^{(0)} \\ 0 & a_0^{(-1)} & & & & & & & \vdots \\ 0 & 0 & \ddots & & & & & & \\ \vdots & \vdots & \ddots & a_0^{(-i+1)} & a_1^{(-i+1)} & \cdots & & & a_{n-i+1}^{(-i+1)} \\ 0 & 0 & \cdots & 0 & a_0^{(-i)} & a_1^{(-i)} & \cdots & & a_{n-i}^{(-i)} \\ a_{-(i+1)}^{(-i)} & 0 & \cdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & & & & & & \\ & & & & & & & \ddots & a_1^{(-i)} \\ a_{-n}^{(-i)} & \cdots & a_{-(i+1)}^{(-i)} & 0 & \cdots & & & 0 & a_0^{(-i)} \end{pmatrix}, \tag{7.2}$$

$$A^{(i)} = \begin{pmatrix} a_0 & 0 & \cdots & & & 0 & a_{i+1}^{(i)} & \cdots & a_n^{(i)} \\ a_{-1}^{(i)} & \ddots & \ddots & & & & & & \vdots \\ \vdots & \ddots & & & & & & \ddots & \\ & & \ddots & \ddots & 0 & \cdots & & 0 & a_{i+1}^{(i)} \\ a_{-n+i}^{(i)} & \cdots & & a_{-1}^{(i)} & a_0 & 0 & \cdots & & 0 \\ a_{-n+i-1}^{(i-1)} & \cdots & & & a_{-1}^{(i-1)} & a_0 & 0 & \cdots & 0 \\ \vdots & \ddots & & & & \ddots & \ddots & \ddots & \vdots \\ & & & & & & & & 0 \\ a_{-n}^{(0)} & \cdots & & & & & & a_{-1}^{(0)} & a_0 \end{pmatrix}. \tag{7.3}$$

The algorithm for updating $A^{(\pm i)}$ is given by:

$$a_j^{(0)} = a_j \ (j = -n, -n+1, \cdots, 0, \cdots, n) \tag{7.4}$$

and for $i = 1, 2, \cdots, n$:

$$a_j^{(-i)} = a_j^{(-i+1)} - \frac{a_{-i}^{(-i+1)}}{a_0} a_{j+i}^{(i-1)} \tag{7.5}$$

$$(j = -n, \cdots, -i-1; 0, \cdots, n-i \text{ for } i < n; j = 0 \text{ for } i = n),$$

$$a_j^{(i)} = a_j^{(i-1)} - \frac{a_i^{(i-1)}}{a_0^{(-i)}} a_{j-i}^{(-i)} \tag{7.6}$$

$$(j = -n+i, \cdots, -1; i+1, \cdots, n \text{ for } i < n).$$

For the case of a symmetric Toeplitz matrix, the above formulae are further simplified. They are noted as equations (3.4a) and (3.4b) in [1], which are

$$a_j^{(0)} = a_{|j|} \ (j = -n, -n+1, \cdots, 0, \cdots, n), \tag{7.7}$$

$$a_j^{(i)} = a_j^{(i-1)} - \frac{a_i^{(i-1)}}{a_0^{(i-1)}} a_{i-j}^{(i-1)}, \tag{7.8}$$

$$a_j^{(-i)} = a_{-j}^{(i)} \ (j = -n+i, \cdots, 0; i+1, \cdots, n \text{ for } i < n; j = 0 \text{ for } i = n). \tag{7.9}$$

Corollary 1 in [1] states that the determinant of A is $a_0 a_0^{(-1)} a_0^{(-2)} \cdots a_0^{(-n)}$. For the case of computing the determinant of a symmetric Toeplitz matrix, we do not need to store $a_j^{(-i)}$. The determinant is calculated as $a_0 a_0^{(1)} a_0^{(2)} \cdots a_0^{(n)} = \prod_{i=0}^{n} a_0^{(i)}$.

My C program named as `DetBareissSym` uses equations (7.7) and (7.8) with modular arithmetic operations. It has the following input and output:

- Input:
  (i) $b = [a_0, \cdots, a_n]$, the entries of a symmetric Toeplitz matrix A of size $(n+1) \times (n+1)$,
  (ii) the integer n,
  (iii) a prime p.
- Output: $\det(A) \bmod p$.

Inside `DetBareissSym`, I used arrays `a`, `a_new`, and `inv`. Both arrays `a` and `a_new` have size $2n+1$ which store the elements $a_j^{(i)}$ and $a_j^{(i+1)}$ for $j = -n, \cdots, n$. The array `inv` has size $n+1$ and it stores the inverses of $a_0^{(i)}$ for $i = 0, \cdots, n$. If any $a_0^{(i)} = 0$, then the program calls `det64s` to compute the determinant using Gaussian elimination instead.

```
LONG DetBareissSym( LONG *b, int n, LONG p ) {
    // Input: vector b=[b_0,...,b_n], diag entries of symmetric Toeplitz
    // matrix A (size (n+1)*(n+1))
    // Output: Det(A) mod p
    LONG *a, *anew, *inv, prod; int i, j;
    if ( b[0]==0 ){ //printf("note: a[0]=0, used G.E.\n");
        LONG *A = matrix64s(n+1); LONG d = tdet64s(b,n+1,p,A); free(A);
        return d;
    }
    a = array( 2*n+1 ); anew = array( 2*n+1 ); inv = array( n+1 );
    a[0] = b[n]; for ( j=1; j<n+1; j++ ){ a[j] = b[n-j]; a[j+n] = b[j]; }
    inv[0] = inv64s( a[n],p ); prod = a[n];
    for ( i=1; i<n; i++ ) {
        for ( j=i; j<n+1; j++ ) { // j=-n+i to 0 in (3.4b)
            anew[j] = sub64s( a[j], mul64s(mul64s(a[n+i],inv[i-1],p),
            a[2*n-j+i], p ), p );
        }
        if ( anew[n]==0 ) { //printf("note:a[%d]=0, used G.E.\n",i);
            LONG *A = matrix64s(n+1); LONG d = tdet64s(b,n+1,p,A); free(A);
            return d;
        }
        inv[i] = inv64s( anew[n],p ); prod = mul64s( anew[n], prod, p );
        for ( j=n+i+1; j<2*n+1; j++ ) { // j=i+1 to n in (3.4b)
            anew[j] = sub64s( a[j], mul64s(mul64s(a[n+i],inv[i-1],p),
            a[2*n-j+i], p ), p );
        }
        for ( j=0;j<2*n+1;j++ ) { a[j] = anew[j]; }
    }
    anew[n] = sub64s( a[n], mul64s(mul64s(a[2*n],inv[n-1],p), a[2*n], p ),
    p );
    return mul64s( anew[n], prod, p );
}
```

7.4 Bivariate dense interpolation

Since the individual degrees of a, $d_i = \deg(a, x_i)$, are pre-computed prior to Hensel lifting, we can use d_1 and d_j as degree bounds for x_1 and x_j to interpolate $A_k(x_1, x_j) = a(x_1, \beta_2^k, \cdots, \beta_{j-1}^k, x_j, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$ (step 8 of Algorithm 13) by using Newton in-

terpolations on both variables x_1 and x_j. Lagrange interpolation also works, I used Newton interpolation. This process is known as *bivariate dense interpolation*.

After $(d_1+1)(d_j+1)$ probes to the black box $\mathbf{B}$, the values

$$M_{l,m} := a(\gamma_l, \beta_2^k, \cdots, \beta_{j-1}^k, \delta_m, \alpha_{j+1}, \cdots, \alpha_n) \bmod p$$

for $0 \le l \le d_1$ and $0 \le m \le d_j$ are stored as a matrix M, as shown in Figure 7.1. Here, $\gamma_l \in \mathbb{Z}_p$ and $\delta_m \in \mathbb{Z}_p$ are evaluation points for x_1 and x_j respectively. $\beta_2, \cdots, \beta_{j-1}$ are chosen from step 2 of Algorithm 13, and $\alpha_{j+1}, \cdots, \alpha_n$ are values from the initial evaluation point $\boldsymbol{\alpha}$ prior to Hensel lifting.

From the columns of M, $A_k(\gamma_l, x_j) \in \mathbb{Z}_p[x_j]$ for $0 \le l \le d_1$ are interpolated using Newton interpolations on the variable x_j. They are denoted as $A_k(\gamma_l, x_j) = c_{l,0} + c_{l,1}x_j + \cdots + c_{l,d_j}x_j^{d_j}$ for $0 \le l \le d_1$. The coefficients $c_{l,m}$ for $0 \le l \le d_1$ and $0 \le m \le d_j$ are stored back onto the matrix M, one column at a time. After updating the last column of M, we get $M_{l,m} = c_{l,m}$.

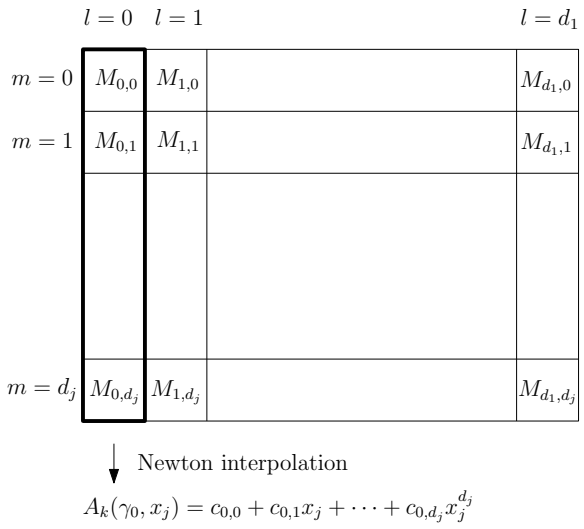

Figure 7.1: The matrix M for storing $(d_1+1)(d_j+1)$ values for bivariate dense interpolation.

Next, the coefficients $c_{l,m}$ for $0 \le l \le d_1$ (the rows of M) are used to interpolate $c_m(x_1) \in \mathbb{Z}_p[x_1]$ for all $0 \le m \le d_j$. $c_m(x_1)$ are the coefficients of $A_k(x_1, x_j) \in \mathbb{Z}_p[x_1][x_j]$. We have

$$A_k(x_1, x_j) = c_0(x_1) + c_1(x_1)x_j + \cdots c_{d_j}(x_1)x_j^{d_j}, \tag{7.10}$$

where $c_m(x_1) = \bar{c}_{0,m} + \bar{c}_{1,m}x_1 + \bar{c}_{d_1,m}x_1^{d_1}$ for $0 \le m \le d_j$. The values $\bar{c}_{l,m}$ are stored back to the matrix M again one row at a time. After updating the last row, we recover the bivariate polynomial $A_k(x_1, x_j)$, with coefficients stored as $M_{l,m} = \bar{c}_{l,m}$.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

We designed and implemented a new black box factorization algorithm for polynomials $a \in \mathbb{Z}[x_1, \cdots, x_n]$. Our algorithm first finds the factors of the primitive part of a, then the factors of the content of a in the main variable x_1.

Two benchmarks for a monic and square-free were presented in Chapter 5. One of them was to compute the factors of $\det(T_n)$, where T_n is a symmetric Toeplitz matrix. I believe that I am the first to compute $\det(T_{16})$ with its factors in the sparse representation. Three more benchmarks were presented in Chapter 6 for the non-monic, non-square-free, and non-primitive cases. For the non-primitive cases, two benchmarks were presented. The first benchmark was to compute $\det(V_n)$, where V_n is a Vandermonde matrix. The second benchmark was to compute the determinant of Dixon matrices, which came from solving polynomial systems of equations. For those two benchmarks, we demonstrated an advantage of our algorithm: when computing the content of a (w.r.t. x_1) is not necessary, the computational cost is significantly saved by computing the factors of the primitive part only. All our timings were much faster than the current best determinant and factorization algorithms in Maple and Magma.

We also presented a complexity analysis for algorithm CMSHL as well as our new black box algorithm CMBBSHL with bounds on failure probabilities. Algorithm CMSHL is Las Vegas, while algorithm CMBBSHL is Monte-Carlo (in fact, Atlantic City). Algorithm CMBBSHL can return a FAIL with a low probability or an incorrect answer (not a FAIL) with a low probability.

For our black box factorization algorithm, we further considered the case for large integer coefficients and coded that in Maple. One benchmark for large integer coefficients is presented in Section 6.5.

8.2 Future work

There are several directions for future research. First, I would like to finish implementing Rubinfeld and Zippel's algorithm [51] (and Kaltofen and Trager's algorithm [31]) for Approach I. We have made an analysis to compare the number of probes to the black box for our algorithm and Rubinfeld and Zippel's algorithm. It will be more convincing to have timings for Rubinfeld and Zippel's algorithm alongside our benchmarks.

Another interesting problem is selecting a large prime from $\mathbb{P}_{63}$ at random, where $\mathbb{P}_{63}$ is the set of all 63-bit primes. In Section 1.6.2, we need to select a prime from $\mathbb{P}_{63}$ at random when computing the total degree of a multivariate polynomial represented by a black box. However, $\mathbb{P}_{63}$ is too big to create explicitly. Because the prime distribution in $\mathbb{P}_{63}$ is not uniform, picking $y \in [2^{62}, 2^{63}]$ at random and using the Maple command `nextprime(y)` does not make the prime selection uniform.

An application of our algorithm that we would like to investigate is solving parametric linear systems of equations. Let A be a matrix of size $m \times m$ with $A_{ij} \in \mathbb{Z}[y_1, \cdots, y_n]$. Let b be a vector of size $m \times 1$ with $b_i \in \mathbb{Z}[y_1, \cdots, y_n]$. Let $A^{(i)}$ be formed from A by replacing the i^{th} column of A with b. We can solve $Ax = b$ by computing $x_i = \det(A^{(i)})/\det(A)$ for $1 \leq i \leq m$ in factored form using a black box factorization algorithm.

Bibliography

[1] E. Bareiss. Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices. *Numerische Mathematik*, **13**(5), 404–424 (1969)

[2] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In Proceedings of STOC '88, pp. 301–309. ACM (1988)

[3] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, **46**(8): 1853–1859, 1967.

[4] L. Bernardin. On Bivariate Hensel Lifting and its Parallelization. In Proceedings of ISSAC '98, pp. 96–100. ACM (1998)

[5] V. Bhargava, S. Saraf and I. Volkovich. Deterministic Factorization of Sparse Polynomials with Bounded Individual Degree. J. ACM 67, 2, Article 8 (April 2020), 28 pages. https://doi.org/10.1145/3365667

[6] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* **18**:478–504, 1971.

[7] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, **36**(154): 587–592, 1981.

[8] T. Chen and M. Monagan. The complexity and parallel implementation of two sparse multivariate Hensel lifting algorithms for polynomial factorization. In Proceedings of CASC 2020, LNCS **12291**: 150–169. Springer (2020)

[9] T. Chen and M. Monagan. Factoring multivariate polynomials represented by black boxes: A Maple + C Implementation. *Math. Comput. Sci.* **16**,18 (2022)

[10] T. Chen and M. Monagan. A new black box factorization algorithm - the non-monic case. In Proceedings of ISSAC 2023. ACM (2023)

[11] S. D. Cohen. The distribution of Galois groups and Hilbert's irreducibility theorem. In Proceedings of the London Mathematical Society (3), 43:227–250 (1981)

[12] K. Connolly. A Maple implementation of FFT-based algorithms for polynomial multipoint evaluation, interpolation, and solving transposed Vandermonde systems. Masters Project, Simon Fraser University, 2020. `http://www.cecm.sfu.ca/CAG/theses/kimberly.pdf`

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. Introduction to Algorithms. MIT Press (2009)

[14] D. Cox, J. Little, D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer (2007)

[15] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. Information Processing Letters. 7 (4): 193–195. (1978)

[16] A. Diaz and E. Kaltofen. FOXBOX: a system for manipulating symbolic objects in black box representation. Proceedings of ISSAC 1998. ACM (1998)

[17] R. Fateman. Comparing the speed of programs for sparse polynomial multiplication. ACM SIGSAM Bulletin 37 (1), pp. 4–15, 2003.

[18] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press (2013)

[19] K. O. Geddes, S. R. Czapor and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Acad. (1992)

[20] M. T. Goodrich and R. Tamassia. *Algorithm Design and Applications*. Wiley (2014)

[21] David Hilbert. Über die Irreducibilität ganzer rational Functionen mit ganzzahigen Koeffizienten. *Algebra · Invariantentheorie · Geometrie*, 264–286. Springer (1933)

[22] M. van Hoeij, M. B. Monagan. A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *Proceedings of ISSAC '2002*, ACM Press, pp. 109–116, 2002.

[23] Q-L. Huang and X-S. Gao. New sparse multivariate polynomial factorization algorithms over integers. In Proceedings of ISSAC 2023. ACM (2023)

[24] J-X. Hu and M. Monagan. A fast parallel sparse polynomial GCD algorithm. *J. Symb. Cmpt.* **105**, 28–63. Elsevier (2021)

[25] A. I. Jinadu. Solving parametric systems using Dixon resultants and sparse interpolation tools. PhD Thesis, Simon Fraser University, 2023.

[26] W. M. Gentleman and S. C. Johnson. Analysis of algorithms, a case study: determinants of matrices with polynomial entries. *ACM Trans. on Math. Soft.* **2**(3): 232–241. ACM (1976)

[27] E. Kaltofen. Sparse Hensel lifting. In Proceedings of EUROCAL '85, LNCS **204**, 4–17. Springer (1985)

[28] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the ACM*, **35**(1), 231–264 (1988)

[29] E. Kaltofen. Computing with polynomials given by straight-line programs II sparse factorization. In Proceedings of the 26th Annual Symposium on Foundations of Computer Science, pp. 450–458. IEEE (1985)

[30] E. Kaltofen. Factorization of polynomials given by straight-line programs. *Adv. Compt. Res.* **5**, 375–412 (1989)

[31] E. Kaltofen E and B. M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Cmpt.* **9**(3), 301–320. Elsevier (1990)

[32] E. Kaltofen and L. Yagati. Improved sparse multivariate polynomial interpolation algorithms. In Proceedings of ISSAC '88, LNCS **358**, 467–474. Springer (1988)

[33] T. Khovanova and Z. Scully. Efficient Calculation of Determinants of Symbolic Matrices with Many Variables, ArXiv: 1304.4691 (2013)

[34] S. Lang. Fundamentals of Diophantine Geometry. Springer-Verlag New York (1983)

[35] G. Lecerf. Improved dense multivariate polynomial factorization algorithms. *J. Symbolic Computation* **42**:477–494 (2007)

[36] W. S. Lee. Early termination strategies in sparse interpolation algorithms. Ph.D. Thesis (2001)

[37] R. A. Mollin. RSA and public-key cryptography. Chapman and Hall/CRC (2003)

[38] M. Monagan. Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In Proceedings of ISSAC '04, pp. 243–249. ACM (2004)

[39] M. Monagan. Linear Hensel lifting for $\mathbb{F}_p[x, y]$ and $\mathbb{Z}[x]$ with cubic cost. In Proceedings of ISSAC 2019, pp. 299–306. ACM (2019)

[40] M. Monagan. Lecture notes on Topics in Computer Algebra. Simon Fraser University, 2019.

[41] M. Monagan. Speeding up polynomial GCD, a crucial operation in Maple. *Maple Transactions.* **2**:1 Article 14457, 2022.

[42] M. Monagan and R. Pearce. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Communications of Computer Algebra*, 48(4): 166-186, December 2014.

[43] M. Monagan and B. Tuncer. Using Sparse Interpolation in Hensel Lifting. In Proceedings of CASC 2016, LNCS **9890**, 381–400. Springer (2016)

[44] M. Monagan and B. Tuncer. Factoring multivariate polynomials with many factors and huge coefficients. In Proceedings of CASC 2018, LNCS **11077**, 319–334. Springer (2018)

[45] M. Monagan and B. Tuncer. Sparse multivariate Hensel lifting: a high-performance design and implementation. In Proceedings of ICMS 2018, LNCS **10931**, 359–368. Springer (2018)

[46] M. Monagan and B. Tuncer. The complexity of sparse Hensel lifting and sparse polynomial factorization. *J. Symb. Cmpt.* **99**, 189–230. Elsevier (2020)

[47] M. Monagan and B. Tuncer. Polynomial factorization in Maple 2019. In Maple in Mathematics Education and Research. *Communications in Computer and Information Science*, **1125**, 341–345. Springer (2020)

[48] M. Monagan and G. Paluck. Linear Hensel lifting for $\mathbb{Z}_p[x, y]$ for n factors with cubic cost. In Proceedings of ISSAC 2022, 169-166. ACM (2022)

[49] G. Paluck. A new bivariate Hensel lifting algorithm for n factors. MSc. Thesis, Simon Fraser University (2019)

[50] D. Roche. What can (and can't) we do with sparse polynomials? In Proceedings of ISSAC 2018, pp. 25–30. ACM (2018)

[51] R. Rubinfeld and R. E. Zippel. A new modular interpolation algorithm for factoring multivariate polynomials. In Proceedings of Algorithmic Number Theory, First International Symposium, ANTS-I (1994)

[52] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, **27**(4), 701–717 (1980)

[53] W. J. Turner. Black box linear algebra with the Linbox library. PhD Thesis, North Carolina State University (2002)

[54] P. S. Wang and L. P. Rothschild. Factoring multivariate polynomials over the integers. *Math. Comp.* **29**, 935–950 (1975)

[55] P. S. Wang. An improved multivariate polynomial factoring algorithm. *Math. Comp.* **32**, 1215–1231 (1978)

[56] P. S. Wang. A p-adic algorithm for univariate partial fractions. In Proceedings of SYMSAC '81, pp. 212–217. ACM (1981)

[57] P. S. Wang, M. J. T. Guy, and J. H. Davenport. p-adic reconstruction of rational numbers. SIGSAM Bulletin, **16**(2) (1982)

[58] D. Y. Y. Yun. The Hensel Lemma in algebraic manipulation. Ph.D. Thesis (1974)

[59] H. Zassenhaus. On Hensel factorization I. *J. Number Theory*, **1**(3), 291–311 (1969)

[60] R. E. Zippel. Probabilistic algorithms for sparse polynomials. LNCS **72**, 216–226 (1979)

[61] R. E. Zippel. Newton's iteration and the sparse Hensel algorithm. In Proceedings of the ACM Symposium on Symbolic Algebraic Computation, pp. 68–72 (1981)

[62] R. E. Zippel. Interpolating polynomials from their values. *J. Symb. Cmpt.* **9**(3), 375–403 (1990)

Appendix A

Maple code to run algorithm CMBBSHL

```
read "CMSHL_NMContLI3.mpl";
read "MakeBlackBoxes.mpl";

# ------------- Initial settings ---------------------------
# ---- Choose a black box mode for creating a black box ----
# 1. BBMPoly: Uses a given polynomial a in Z[x1,...,xn] to generate a
#             modular black box Bm(alpha,p)=a(alpha) mod p.
# 2. BBdet: Uses a matrix A with entries in Z[x1,...,xn] to create a
#            modular black box Bm(alpha,p)=det(A(alpha)) mod p.
BBMode := BBMPoly:

# --- For BBdet ONLY ---
# BBdetCode = C: both BBeval and BBdet coded in C, for general (non-
# structured) matrices
# BBdetCode = Maple: both BBeval and BBdet coded in Maple.
BBdetCode := C:

# ---- Choose a test example ----
# BBMPoly: testex = 0,1,2,3;
# BBdet: testex = 0,1,...,9.
testex := 2:

# --------------- For CMBBSHL ---------------
# ---- Choose p and Ntilde ----
p := prevprime(2^62-1):
Ntilde := 4001: # Ntilde as in our ISSAC 2023 paper

# ---- Options for content and square-free computation ----
Cont_Flag := 1: # 0 or 1 for content computation in a chosen variable
sqfinterp := 0: # 0 or 1 for an option to use SquareFreeImage
```

```
# ---- Choose Maple or C code for each subroutine ----
# Subroutines: 1. Bivariate interp, 2. Eval fjm1,
#              3. BHL, 4. Vandermonde Solves.
# For large integer coefficients, use Maple code for all.
MapleCode := [C,C,C,C]:
# ------------- End of initial settings ---------------------

C := 0: Maple := 1:
LI := 0: # for large integer coeffs
printf("Cont_Flag=%d, sqfinterp=%d, MapleCode=%a, LargeInt=%a\n", Cont_Flag,
sqfinterp, MapleCode, LI);
randzp := rand(p):
randzp1 := rand(Ntilde):

# Read in data and create black boxes
if BBMode = `BBMPoly` then
    # Make a black box from a given polynomial a in Z[x1,...,xn]
    if testex = 0 then a := 232;
    elif testex = 1 then
        a := 299*(x^2+x+1)*(3*x+19);
    elif testex = 2 then
        a := (x1+x2+x3+1)*(x2+x3+1);
    elif testex = 3 then
        a := (x1+x3+3*x4^2+2)*(31*x1^2*x3+x2+3*x4+1)*(x3+2)*(x2+1)^2*(x4+3);
    fi;
    X := convert(indets(a),list); printf("a = %a, X = %a\n",a,X);
    N := nops(X);
    # ---- OPTIONAL: Choose a variable permutation ----
    # VarPermGen(N,opt,MainVar)
    # opt = 0, no perm; opt = 1, reverse; opt = 2, random perm;
    # opt > 2, choose a MainVar between 1 to N, swap with the first variable.
    VPL := VarPermGen( N, 3, 2 );
    # -------------------------------------------------
    Var_Perm := VPL[1]; # indicator for variable permutation, 0 or 1.
    VP := VPL[2]; # A list of variable permutation.
    Xnew := Array(1..N);
    for i to N do Xnew[i] := X[VP[i]]; od;
    Xnew := convert(Xnew,list);
    BBInput := MakeBBMPoly( a, VP ); # Outputs procedure BB_MPoly(alpha,p)
elif BBMode = `BBdet` then
    # Make a black box for det(A)
    # ----- Choose a matrix for testex 0,1,2,8,9 ONLY -----
    # --- For testex 0 ----------
    n0 := 4: # The size of the square matrix. For Toeplitz, need n0>=1.
    # --- For testex 1,2,8,9: B||m, T||m, TL||m, BL||m
    # m := 5;
```

```
    # --- For testex 6 (robotarms): m = 1,2,3,4 for b1,b2,t1,t2 ---
    m := 9;
    # ------------------------------------------------------
    if testex = 0 then
        #A := LinearAlgebra:-RandomMatrix(n0,n0,generator=-1000..1000);
        v := <seq(x||i,i=1..n0)>;
        A := LinearAlgebra:-VandermondeMatrix(v);
        #A := LinearAlgebra:-ToeplitzMatrix(v,symmetric);
    elif testex = 1 then
        reader := subs(FILENAME=B||m, proc() read FILENAME; end); reader();
    elif testex = 2 then
        reader := subs(FILENAME=T||m, proc() read FILENAME; end); reader();
        A := T||m;
    elif testex = 3 then
        reader := subs(FILENAME=heron3dB, proc() read FILENAME; end);
        reader();
        A := B;
    elif testex = 4 then
        reader := subs(FILENAME=heron4dB, proc() read FILENAME; end);
        reader();
        A := B;
    elif testex = 5 then
        reader := subs(FILENAME=heron5dB, proc() read FILENAME; end);
        reader();
        A := B;
    elif testex = 6 then
        if m = 1 then ReadFileName := robotarmsB_b1;
        elif m = 2 then ReadFileName := robotarmsB_b2;
        elif m = 3 then ReadFileName := robotarmsB_t1;
        elif m = 4 then ReadFileName := robotarmsB_t2;
        fi;
        reader := subs(FILENAME=ReadFileName, proc() read FILENAME; end);
        reader();
        A := B;
    elif testex = 7 then
        reader := subs(FILENAME="LinearSystem.mpl", proc() read FILENAME;
end);
        reader();
    elif testex = 8 then # Large Interger case TL4-TL9: p is read-in
        reader := subs(FILENAME=TLI||m, proc() read FILENAME; end); reader();
        A := TL||m;
        LI := 1;
    elif testex = 9 then # Large Integer case BL4-BL9: p is read-in
        reader := subs(FILENAME=BLI||m, proc() read FILENAME; end); reader();
        p := nextprime(p^2);
        A := BL||m;
        LI := 1;
```

```
    fi;
    printf("A = %a\n",A);
    X := convert(indets(A),list);
    N := nops(X);
    n := LinearAlgebra:-RowDimension(A);
    # ---- OPTIONAL: Choose a variable permutation ----
    # VarPermGen(N,opt,MainVar)
    # opt = 0, no perm; opt = 1, reverse; opt = 2, random perm;
    # opt > 2, choose a MainVar between 1 to N, swap with the first variable.
    VPL := VarPermGen( N, 0, 1 );
    #VPL := [1,[6,2,3,4,1,5,7,8,9,10,11]];
    # ------------------------------------------------
    Var_Perm := VPL[1]; # indicator for variable permutation, 0 or 1.
    VP := VPL[2]; # List of variable permutation.
    Xnew := Array(1..N);
    for i to N do Xnew[i] := X[VP[i]]; od;
    Xnew := convert(Xnew,list);
    if BBdetCode = `C` then
        BBInput := MakeBBdet_C( A, VP );
    elif BBdetCode = `Maple` then
        BBInput := MakeBBdet_Maple( A, VP, 2 ); # The 3rd arg = 2: Use G.E.
    fi;
fi:

# Select alpha's at random:
if LI = 0 then alpha := Array([seq(randzp1(),i=1..N)],datatype=integer[8]);
else alpha := Array([seq(randzp1(),i=1..N)]);
fi:

printf("No.of variables = %d\n", N);
printf("    X = %a\n Xnew = %a\n",X, Xnew);
printf("Variables permutated? %a\n Variable perm = %a\n", VPL[1], VP);
printf("Large Integer? LI = %d\n", LI);
printf("alpha = %a\n",alpha);

# Compute a total degree bound
if BBMode = `BBdet` then tdegbd := 0:
    for ii to n do maxdeg[ii] := -1;
        for jj to n do dd := degree(A[ii,jj]);
            if dd > maxdeg[ii] then maxdeg[ii] := dd; fi;
        od;
        tdegbd += maxdeg[ii];
    od:
else tdegbd := degree(a);
fi:
printf("total deg bound = %a\n", tdegbd);
```

```
# Compute deg(a,xj) w.h.p.
CNT := 0: tBBeval := 0: tBBdet := 0:
st := time():
for j to N do #printf("j = %d\n",j);
    deg[j] := degB(BBInput,N,j,p,tdegbd,LI);
od:
et := time()-st: printf("time for degree computation = %f\n", et);
degA := [seq(deg[j],j=1..N)]:
printf("degA = %a\n", degA);
printf("no.of probes for deg compt = %d\n", CNT):
printf("total times for BB in deg compt: BBeval = %f, BBdet = %f\n",
tBBeval,tBBdet);

# Start CMBBSHL
CNT := 0: tBBeval := 0: tBBdet := 0:
st := time():
ff := CMBBSHLcont( BBInput, Xnew, alpha, degA, N, p, Var_Perm, Cont_Flag,
sqfinterp, MapleCode, LI );
et := time() - st:
printf("Total time for CMBBSHL = %f\n", et);
printf("Total no.of probes for CMBBSHL = %d\n", CNT):

saver := subs(FILENAME=factors_ContNMLI||BBMode||testex||m, proc() save ff,
FILENAME; end):
saver():
```