Factoring Multivariate Polynomials Represented by Black Boxes – A Maple + C Implementation

Tian Chen and Michael Monagan

Department of Mathematics, Simon Fraser University,
Burnaby, British Columbia, V5A 1S6, CANADA
tca71@sfu.ca, mmonagan@sfu.ca

Abstract. Our goal is to develop fast parallel algorithms to factor multivariate polynomials with integer coefficients. The authors contributed a parallel sparse Hensel lifting algorithm (CMSHL) to factor multivariate polynomials in their sparse representation (Chen and Monagan, 2020). The dominating cost of CMSHL is polynomial evaluations.

To reduce this cost, in this work we represent the polynomial to be factored by a black box. We give an algorithm that computes the factors of the polynomial in their sparse representation directly using our modified CMSHL algorithm. Our new algorithm requires fewer probes to the black box than the algorithm of Kaltofen and Trager from 1990.

We have implemented our algorithm in Maple with major subroutines coded in C. We present timing benchmarks for two examples of factoring determinants of matrices of polynomials. Our algorithm is much faster than Maple's best determinant and factor algorithms on our benchmarks. We are able to compute the factors of $\det T_n$ for $n = 16$ where T_n is the $n \times n$ symmetric Toeplitz matrix with symbolic entries $x_1, x_2, \ldots, x_n$.

Keywords: Sparse Hensel Lifting, Multivariate Polynomial Factorization, Black Box Representation, Determinant Algorithms

1 Introduction

Polynomial factorization has been a central topic in computer algebra. It has applications in diverse fields such as algebraic coding theory, cryptography, number theory, algebraic geometry and biological modelling [12]. Our work focuses on the design and implementation of algorithms to factor multivariate polynomials with integer coefficients. In 2020, Chen and Monagan [6] developed a highly parallelizable sparse Hensel lifting algorithm (CMSHL) to factor multivariate polynomials input in the *sparse representation*. The dominating cost of CMSHL is evaluating the input polynomial $a(x_1, \ldots, x_n)$ at many points. To reduce this cost, in this work, we represent the polynomial to be factored by a *black box* and aim to compute its factors in their *sparse representation*. An example is to factor 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. We reduce the cost of evaluating a as well as the memory space needed to store a in its sparse representation.

To factor a multivariate polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$, Yun [35] and Wang [34] initially developed multivariate Hensel lifting (MHL). MHL first chooses integers $\alpha_2, \cdots, \alpha_n$ and factors the univariate image $a(x_1, \alpha_2, \cdots, \alpha_n)$ in $\mathbb{Z}[x_1]$. Then it recovers the factors one variable at a time (see Chap. 6 of [12] for a detailed description). Wang's MHL has been implemented in many computer algebra systems including Maple, Magma, Macsyma, Mathematica and Singular, and it is still widely used today.

However, when factors are sparse and the evaluation points $\alpha_2, \cdots, \alpha_n$ are mostly non-zero, Wang's MHL algorithm can be exponential in the number of variables [26]. To overcome this, Zippel [37] introduced sparse Hensel lifting (SHL) algorithm in 1981 which was the first random polynomial time algorithm that takes sparsity into account. Kaltofen [16] presented another SHL algorithm in 1985. In 2016, Mongan and Tuncer [26] developed a new sparse Hensel lifting algorithm MTSHL. It solves the multivariate Diophantine equations that appear in Wang's MHL in random polynomial time. MTSHL was integrated into Maple 2019 [30]. In 2018, Monagan and Tuncer [28] gave another algorithm which is more suitable for parallelization. In 2020, the authors presented the algorithm CMSHL [6]. It is highly parallelizable as it does not do any multivariate polynomial arithmetic and it has eliminated a long standing problem of expression swell.

If the polynomial a is dense then alternative methods for factoring should be used. We cite the work of Lecerf [22] who factors a in $\mathbb{Q}[[x_2, \ldots, x_n]][x_1]$ and uses fast arithmetic for power series in $\mathbb{Q}[[x_2, \ldots, x_n]]$.

The dominating cost of CMSHL is evaluating the input polynomial $a \in \mathbb{Z}[x_1, \cdots, x_n]$ at many points (see Sect. 4 of [6]). To reduce this cost, we represent the polynomial to be factored by a *black box*. The goal is to compute the factors in their *sparse representation*. The *sparse* and the *black box representations* are defined in Sect. 2.1.

In 1990, Kaltofen and Trager [18] contributed the first black box factorization algorithm for multivariate polynomials. Given an evaluation point of the input polynomial with coefficients in a field, their factorization algorithm outputs an evaluation of each factor. The sparse representation of the factors can then be recovered using *sparse polynomial interpolation*. Early references for sparse polynomial interpolation include [3, 19, 38]. For a recent bibliography we refer the reader to Roche [32]. Kaltofen and Trager's algorithm was implemented in FOXBOX [9] which was written in C++.

As far as we know, there has not been any black box factorization algorithm developed since Kaltofen and Trager [18] in 1990. Instead of first getting the evaluations of the factors and then doing a sparse interpolation as in [18], our new black box factorization algorithm outputs the factors in sparse representation directly via the modified algorithm CMSHL. Our new algorithm requires fewer probes to the black box than Kaltofen and Trager's algorithm (see Sect. 2.3). We have made a hybrid Maple + C implementation for our new algorithm. The main program is written in Maple (the user has the option to input a symbolic

black box, e.g. a matrix with multivariate polynomial entries) and the major subroutines are coded in C to speed up computations.

This paper is organized as follows. Sect. 2 describes two existing algorithms to tackle the black box factorization problem and our new algorithm (modified CMSHL). We have modified algorithm CMSHL in [6] to incorporate the black box and extended it to compute multi-factors (still monic). Sect. 3 describes the implementation details of the subroutines that were carefully coded in C. In Sect. 4 we present timing benchmarks for two examples of factoring determinants of matrices of polynomials. Our timing benchmarks show that our new algorithm is much faster than Maple's current best determinant and factor algorithms. We were able to compute the factors of $\det(T_n)$ for a symmetric Toeplitz matrix T_n with symbolic entries $x_1, x_2, \ldots, x_n$ for $n = 16$. To our knowledge, this has never been computed before. Sect. 5 describes several algorithms for computing the determinants of symbolic matrices in the sparse representation. All run out of space when computing $\det(T_{16})$. We contribute a space improvement for the Gentleman-Johnson algorithm [14] so that we could compute $\det(T_{15})$ in Maple. Sect. 6 is a conclusion.

2 Algorithms for Black Box Factorization

2.1 The sparse and the black box representation of a polynomial

The *sparse representation* of a polynomial $f \in \mathbb{Z}[x_1, \cdots, x_n]$ consists of a list of coefficients c_k and exponents $(e_{k_1}, \cdots, e_{k_n})$ such that $f = \sum_{k=1}^{t} c_k \cdot x_1^{e_{k_1}} \cdots x_n^{e_{k_n}}$, $c_k \in \mathbb{Z}$, where $c_k \neq 0$ and t is the number of non-zero terms of f. It is a natural, readable and *explicit* representation (Sect. 16.6 of [11]). On the other hand, the *black box representation* of a polynomial $f \in \mathbb{Z}[x_1, \cdots, x_n]$ is a program which takes inputs a prime p and an evaluation point $\boldsymbol{\alpha} \in \mathbb{Z}_p^n$ and outputs $f(\boldsymbol{\alpha}) \bmod p$ (Figure 1). It is one of the most space efficient *implicit* representations [18].

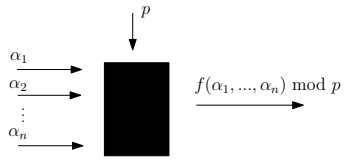

Fig. 1. The black box representation of $f \in \mathbb{Z}[x_1, ..., x_n]$.

Given a black box representation for a polynomial a, what information can we determine about it? For our factorization algorithm, we need to know the degrees, $\deg(a, x_i)$, for each variable x_i, $i = 1, \cdots, n$. To compute $\deg(a, x_i)$, we first pick $\alpha_1, \cdots, \alpha_n \in \mathbb{Z}_p$ at random and define $h(v) := a(\alpha_1, \cdots, \alpha_{i-1}, v, \alpha_{i+1}, \cdots, \alpha_n)$. Theorem 1 below says that if p is large, then $\deg(h, v) = \deg(a, x_i)$ with high probability.

Theorem 1.

$$\text{Prob}[\deg(h,v) \neq \deg(a,x_i)] \leq \frac{\deg(a) - \deg(a,x_i)}{p}. \tag{1}$$

The proof of Theorem 1 uses the Schwartz-Zippel Lemma [33, 36]:

Lemma 1.
Let D be an integral domain and let $f \in D[x_1, ..., x_n]$ such that $f \neq 0$. Let $S \subseteq D$ such that $|S| = k$, $k \in \mathbb{Z}^+$, and let $\beta \in S^n$ be chosen at random, then

$$\text{Prob}[f(\beta) = 0] \leq \frac{d}{|S|}, \text{ where } d = \deg(f). \tag{2}$$

Proof of Theorem 1.
Let $d_i = \deg(a, x_i)$ and let $a = \sum_{j=0}^{d_i} q_j(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)x_i^j$.
For example, $a = (96x_2 + 5)x_1 + (25x_2^3 + 91x_3^3 - 17x_2x_3)$ where $\deg(a, x_1) = 1$.
We have $q_1 = 96x_2 + 5$ and $q_2 = 25x_2^3 + 91x_3^3 - 17x_2x_3$.
Now,

$$\begin{aligned} h(v) := a(v, \alpha_2, \alpha_3) &= (96\alpha_2 + 5)v + (25\alpha_2^3 + 91\alpha_3^3 - 17\alpha_2\alpha_3) \\ &= q_1(\alpha_2, \alpha_3)v + q_0(\alpha_2, \alpha_3). \end{aligned}$$

We can see that $\deg(h, v) \neq \deg(a, x_1) \iff q_1(\alpha_2, \alpha_3) = 0$.
In general, by Schwartz-Zippel Lemma,

$$\begin{aligned} \text{Prob}[\deg(h,v) \neq \deg(a,x_i)] &= \text{Prob}[q_{d_i}(\alpha_1, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_n) = 0] \\ &\leq \frac{\deg(a) - d_i}{p}. \end{aligned}$$

□

If we know a degree bound $d \geq \deg(a, x_i)$ then we may interpolate $h(v)$ using any points $v_0, v_1, \ldots, v_d$ in $\mathbb{Z}_p$, that is, we interpolate $h(v)$ from $(v_k, h(v_k))$, $k = 0, 1, \cdots, d$. If we do not have such a degree bound then, we interpolate $h(v)$ using random points in $\mathbb{Z}_p$ to determine $\deg(h)$ with high probability.

2.2 Modified algorithm CMSHL for the black box

Let $a \in \mathbb{Z}[x_1, \cdots, x_n]$ be represented by a black box **B**. Three ways to compute its factors in the sparse representation are shown in Figure 2. Method 0 first interpolates the sparse representation of a then factors it using a sparse Hensel lifting algorithm [6]. Method I adapts Kaltofen and Trager's algorithm [18] to first get black boxes for the factors and then performs sparse interpolation. We contribute Method II which outputs the factors in their sparse representation directly using our modified CMSHL algorithm.

If the input polynomial a is represented in its sparse form, a few preliminary steps are done before sparse Hensel lifting [6, 29]. The first step is to remove

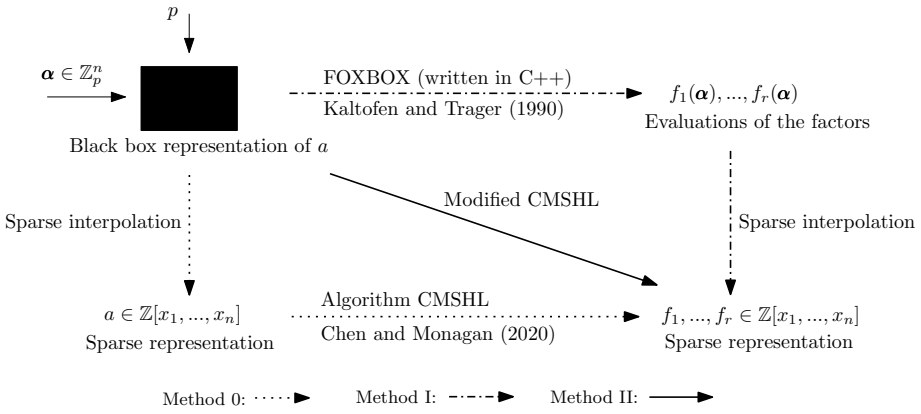

Fig. 2. Factoring $a \in \mathbb{Z}[x_1, ..., x_n]$ given by a black box.

the content of a in a chosen variable, say x_1. For $a = \sum_{i=0}^{d} a_i(x_2, \cdots, x_n)x_1^i$, the content of a is $\gcd(a_0, \cdots, a_d)$, a polynomial with one fewer variables which can be factored recursively. The second step is to identify any repeated factors in a by doing a *square-free factorization* (see Chap. 8 of [12]). After this, let $a = f_1 f_2 \cdots f_r$ denote the irreducible factorization of a over $\mathbb{Z}$. Thirdly, an evaluation point $\boldsymbol{\alpha} = (\alpha_2, \cdots, \alpha_n)$ is chosen and then $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}$. From Hilbert's irreducibility theorem [13, 20], the evaluated factors $f_\rho(x_1, \boldsymbol{\alpha})$, $\rho = 1, \cdots, r$, remain irreducible over $\mathbb{Z}$ with high probability. Finally, sparse Hensel lifting is performed to recover the factors one variable at a time.

If a is represented by a black box, the order of the above steps is different. One way is to first make a hence all the factors of a monic, in x_1 say. Then an evaluation point $\boldsymbol{\alpha}$ is chosen and $a(x_1, \boldsymbol{\alpha})$ is factored over $\mathbb{Z}$. The square-free factors can be identified at this point. After sparse Hensel lifting, the leading coefficients (as well as the content) can be recovered at the end.

For simplicity, in this paper, we only consider irreducible factors that are monic in x_1. We modified algorithm CMSHL in [6] to replace the evaluations of a with probes to the black box. Let $f_{\rho,j}$ denotes $f_\rho(x_1, \cdots, x_j, \alpha_{j+1}, \cdots, \alpha_n)$, $\rho = 1, \cdots, r$. The input to CMSHL is a, $f_{1,1}, f_{2,1}, \cdots, f_{r,1}$, $\boldsymbol{\alpha}$ and a prime p such that $\gcd(f_{k,1}, f_{l,1}) = 1$ for $1 \le k, l \le r$ in $\mathbb{Z}_p[x_1]$. Algorithm CMSHL lifts $f_{1,1}, f_{2,1}, \cdots, f_{r,1}$ to $f_{1,2}, f_{2,2}, \cdots, f_{r,2}$, then lifts $f_{1,2}, f_{2,2}, \cdots, f_{r,2}$ to $f_{1,3}, f_{2,3}$, $\cdots$, $f_{r,3}$ etc. After the j^{th} step of Hensel lifting we have $a_j = \prod f_{\rho,j} \bmod p$ so that at the end, $a_n = \prod f_{\rho,n} \bmod p$. To recover the integer coefficients, one may choose a sufficient large p, or, if necessary, do a subsequent p-adic lift [27].

The j^{th} Hensel lifting step is shown in Algorithm 1. There are 4 major steps, namely (1) the probes to the black box to interpolate the bivariate images $A_k(x_1, x_j)$ of a in step 10, (2) evaluating the factors $f_{i,j-1}$ for $1 \le i \le r$ in step 11, (3) the bivariate Hensel lifts (see [5, 31]) in step 13, and (4) solving the Vandermonde systems (see [38]) in step 19. In order to get the best performance we have coded each of these four steps in C. The number of arithmetic opera-

Algorithm 1 CMSHL for black box: Hensel lifting $\mathsf{x_j}$ (multi-factors).

Input: A prime p, $\alpha_\mathsf{j} \in \mathbb{Z}_\mathsf{p}$, $\mathbf{B}$ (black box representation of a monic in $\mathsf{x_1}$), $\mathsf{f_{1,j-1}}, \cdots, \mathsf{f_{r,j-1}} \in \mathbb{Z}_\mathsf{p}[\mathsf{x_1}, \cdots, \mathsf{x_{j-1}}]$ s.t. $\mathsf{a_j}(\mathsf{x_j} = \alpha_\mathsf{j}) = \prod_{\rho=1}^{\mathsf{r}} \mathsf{f}_{\rho,\mathsf{j-1}}$ with $\mathsf{j} > 2$.
Output: $\mathsf{f_{1,j}}, \cdots, \mathsf{f_{r,j}} \in \mathbb{Z}_\mathsf{p}[\mathsf{x_1}, \cdots, \mathsf{x_j}]$ s.t. $\mathsf{a_j} = \prod_{\rho=1}^{\mathsf{r}} \mathsf{f}_{\rho,\mathsf{j}}$ where $\mathsf{f}_{\rho,\mathsf{j}}(\mathsf{x_j}=\alpha_\mathsf{j}) = \mathsf{f}_{\rho,\mathsf{j-1}}$ for $1 \le \rho \le \mathsf{r}$; Otherwise, **return** FAIL.
Let $\mathsf{f}_{\rho,\mathsf{j-1}} = \mathsf{x_1}^{\mathsf{df}_\rho} + \sum_{\mathsf{i}=0}^{\mathsf{df}_\rho - 1} \sigma_{\rho,\mathsf{i}}(\mathsf{x_2}, ..., \mathsf{x_{j-1}})\mathsf{x_1^i}$ where $\sigma_{\rho,\mathsf{i}} = \sum_{\mathsf{k}=1}^{\mathsf{s}_{\rho,\mathsf{i}}} \mathsf{c}_{\rho,\mathsf{ik}}\mathsf{M}_{\rho,\mathsf{ik}}$ and $\mathsf{M}_{\rho,\mathsf{ik}}$ are the monomials in $\sigma_{\rho,\mathsf{i}}$ for $1 \le \rho \le \mathsf{r}$.
Pick $\boldsymbol{\beta} = (\beta_2, \cdots, \beta_{\mathsf{j-1}}) \in \mathbb{Z}_\mathsf{p}^{\mathsf{j}-2}$ at random.
Evaluate: $\{\mathcal{S}_\rho = \{\mathsf{S}_{\rho,\mathsf{i}} = \{\mathsf{m}_{\rho,\mathsf{ik}} = \mathsf{M}_{\rho,\mathsf{ik}}(\boldsymbol{\beta}), 1 \le \mathsf{k} \le \mathsf{s}_{\rho,\mathsf{i}}\}, 0 \le \mathsf{i} \le \mathsf{df}_\rho - 1\}, 1 \le \rho \le \mathsf{r}\}$.
if any $|\mathsf{S}_{\rho,\mathsf{i}}| \ne \mathsf{s}_{\rho,\mathsf{i}}$ **then return** FAIL **end if**
Let s be the maximum of $\mathsf{s}_{\rho,\mathsf{i}}$.
for k from 1 to s **in parallel do**
  Let $\mathsf{Y_k} = (\mathsf{x_2} = \beta_2^\mathsf{k}, \cdots, \mathsf{x_{j-1}} = \beta_{\mathsf{j-1}}^\mathsf{k})$.
  $\mathsf{A_k} \leftarrow \mathsf{a_j}(\mathsf{x_1}, \mathsf{Y_k}, \mathsf{x_j})$. // via probes to $\mathbf{B}$ and interpolation . $\mathcal{O}(\mathsf{sd_1d_j} \cdot C(\text{probe } \mathbf{B}))$
  $\mathsf{F_{1,k}}, \cdots, \mathsf{F_{r,k}} \leftarrow \mathsf{f_{1,j-1}}(\mathsf{x_1}, \mathsf{Y_k}), \cdots, \mathsf{f_{r,j-1}}(\mathsf{x_1}, \mathsf{Y_k})$. ..... $\mathcal{O}(\mathsf{s}(\#\mathsf{f_1} + \cdots + \#\mathsf{f_r}))$
  if $\gcd(\mathsf{F}_{\rho,\mathsf{k}}, \mathsf{F}_{\phi,\mathsf{k}}) \ne 1$ for any $\rho \ne \phi$ $(1 \le \rho, \phi \le \mathsf{r})$ **then return** FAIL **end if**
  $\mathsf{f_{1,k}}, \cdots \mathsf{f_{r,k}} \leftarrow$ *BivariateHenselLift*$(\mathsf{A_k}, \mathsf{F_{1,k}}, \cdots, \mathsf{F_{r,k}}, \alpha_\mathsf{j}, \mathsf{p})$. .. $\mathcal{O}(\mathsf{sr}(\mathsf{d_1d_j^2} + \mathsf{d_1^2d_j}))$
end for
Let $\mathsf{f}_{\rho,\mathsf{k}} = \mathsf{x_1}^{\mathsf{df}_\rho} + \sum_{\mathsf{l}=1}^{\mu_\rho} \alpha_{\rho,\mathsf{kl}}\tilde{\mathsf{M}}_{\rho,\mathsf{l}}(\mathsf{x_1}, \mathsf{x_j})$ for $1 \le \mathsf{k} \le \mathsf{s}$ where $\mu_\rho \le \mathsf{d_1d_j}$ for $1 \le \rho \le \mathsf{r}$.
for ρ from 1 to r **do**
  for l from 1 to μ_ρ **in parallel do**
    $\mathsf{i} \leftarrow \deg(\tilde{\mathsf{M}}_{\rho,\mathsf{l}}, \mathsf{x_1})$.
    Solve the linear system for $\mathsf{c}_{\rho,\mathsf{lk}}$: $\{\sum_{\mathsf{k}=1}^{\mathsf{s}_{\rho,\mathsf{i}}} \mathsf{m}_{\rho,\mathsf{ik}}^\mathsf{n}\mathsf{c}_{\rho,\mathsf{lk}} = \alpha_{\rho,\mathsf{nl}} \text{ for } 1 \le \mathsf{n} \le \mathsf{s}_{\rho,\mathsf{i}}\}$.
  end for..... $\mathcal{O}(\mathsf{sd_j}(\#\mathsf{f_1} + \cdots + \#\mathsf{f_r}))$
  Construct $\mathsf{f}_{\rho,\mathsf{j}} \leftarrow \mathsf{x_1}^{\mathsf{df}_\rho} + \sum_{\mathsf{l}=1}^{\mu_\rho} \left(\sum_{\mathsf{k}=1}^{\mathsf{s}_{\rho,\mathsf{i}}} \mathsf{c}_{\rho,\mathsf{lk}}\mathsf{M}_{\rho,\mathsf{ik}}(\mathsf{x_2}, ..., \mathsf{x_{j-1}})\right) \tilde{\mathsf{M}}_{\rho,\mathsf{l}}(\mathsf{x_1}, \mathsf{x_j})$.
end for
Pick $\boldsymbol{\beta} \in \mathbb{Z}_p^j$ at random.
if $\mathbf{B}(\boldsymbol{\beta}, \alpha_{j+1}, \cdots, \alpha_n) = \prod_{\rho=1}^{r} f_{\rho,j}(\boldsymbol{\beta})$ **then return** $\mathsf{f_{1,j}} \cdots, \mathsf{f_{r,j}}$ **else return** FAIL **end if**

tions in $\mathbb{Z}_p$ for these steps is shown in blue. $\#f$ denotes the number of terms in a polynomial f. Note that all four steps are parallelizable.

2.3 The number of probes to the black box

To analyze black box algorithms we are mostly interested in the number of probes to the black box (the number of calls to the program $\mathbf{B}$). The following estimates show that our new algorithm (Method II) requires fewer probes to the black box than Method I since $s < \#f_{\max}$ [6]. The quantity s is the number of bivariate images needed in algorithm CMSHL which is $\max \#\sigma_{\rho,i}$ where the $f_{\rho,n} = x_1^d + \sum_{i=0}^{d-1} \sigma_{\rho,i}(x_2, \ldots, x_n)x_1^i$ are the factors of a. The number $\#f_{\max}$ is the maximum number of terms of the factors of a.

- Method I: $O(nd_1d_{\max}\#f_{\max})$ probes to the black box $\mathbf{B}$, plus $O(nd_{\max}\#f_{\max})$ univariate polynomial factorizations.
- Method II: $O(nd_1d_{\max}s)$ probes to the black box $\mathbf{B}$, plus one univariate polynomial factorization in total.

In the above, n is the number of variables of a, $d_1 = \deg(a, x_1)$, $d_{\max} = \max_{1 \le j \le n}(d_j)$. For Method I, since we factor polynomials with coefficients in $\mathbb{Z}$, we use integer substitutions for each variable $x_1, \ldots, x_n$ and factor univariate polynomials instead of using linear substitutions and factoring bivariate polynomials as in [18] (the polynomial to be factored in [18] has coefficients in a field). By the Hilbert irreducibility theorem, the resulting univariate polynomial has the same number of factors as a with high probability. Zippel's sparse interpolation requires $O(ndt)$ probes to the black box of a polynomial with n variables, total degree d and t non-zero terms. To obtain each evaluation of the factors, we need $O(d_1)$ probes to the black box $\mathbf{B}$ and interpolation to get a univariate image in x_1 plus a univariate factorization. Thus in total we have $O(nd_1d_{\max}\#f_{\max})$ number of probes to the black box $\mathbf{B}$ and $O(nd_{\max}\#f_{\max})$ univariate polynomial factorizations.

For Method II, Step 10 of Algorithm 1 is the only step that probes the black box $\mathbf{B}$. Using dense interpolation 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, hence the total number of probes is $O(nd_1d_{\max}s)$. And there is only one univariate polynomial factorization needed in total, at the initial stage, i.e. before CMSHL starts.

As an example consider factoring the determinant of T_n the $n \times n$ symmetric Toeplitz matrix shown in Figure 3. The determinant $\det T_n$ has two factors f_1

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

Fig. 3. The symmetric Toeplitz matrix T_n.

and f_2. Table 1 shows the number of terms of $\det T_n$, f_1, f_2 and the parameter s. Here s is coefficient of degree 0 in x_1 of the larger factor f_1. The data shows that the number of terms of $\det T_n$ is much larger than the largest factor which justifies the black box approach. Also s is smaller than $\max(\#f_1, \#f_2)$. Also shown in Table 1 is $\tilde{s}$ which is the number of trivariate images that would be needed to interpolate the factors from trivariate images in $\mathbb{Z}_p[x_1, x_2, x_j]$ instead of bivariate images in $\mathbb{Z}_p[x_1, x_j]$. Since $\tilde{s} < s$ we might get a speedup if we did that.

The cost of each probe to the black box, C(probe $\mathbf{B}$), differs from case to case. For Method II, the dominating cost is either probes to the black box (step 10) or, if the factors have a large number of terms, solving Vandermonde systems (step 19) (see Theorem 4 in [6] for a detailed complexity analysis of algorithm CMSHL). We present both cases (benchmarks 1 and 2) in Sect. 4. Bivariate Hensel lifts (step 13) do not dominate unless the degrees $\deg(a, x_i)$ are high.

n	$\#\det(T_n)$	$\#f_1,\ \#f_2$	$\tilde{s}$	s
8	1628	$167, 167$	38	93
9	6090	$294, 153$	50	86
10	23797	$931, 931$	229	522
11	90296	$1730, 849$	337	814
12	350726	$5579, 5579$	1465	3174
13	1338076	$10611, 4983$	2297	5223
14	5165957	$34937, 34937$	9705	19960
15	19732508	66684, 30458	15712	34081
16	-	221854, 221854	64524	127690

Table 1. Number of terms of $\det(T_n)$ and its factors f_1 and f_2.

If the dominating cost is probes to the black box, Method II is certainly more efficient than Method I. If the dominating cost is solving Vandermonde systems (due to a large s, the number of bivariate images needed), we could try using trivariate images instead.

3 Implementation

We have made a hybrid Maple + C implementation for our new algorithm. The main program is coded in Maple and all major subroutines are coded in 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. The four main steps coded in C are the following.

Step 10 interpolates $A_k(x_1, x_j) = a_j(x_1, Y_k, x_j)$ by probing the black box at points. The black box $\mathbf{B}$ can be Maple code or C code.

Step 11 evaluates the factors at Y_k to obtain univariate images. This step, which initially was a bottleneck, is described in Sect 3.1.

For the bivariate Hensel lifts in Step 13 we use the new cubic algorithm of Monagan and Paluck [24, 31] that costs $O(d_1 d_j^2 + d_1^2 d_j)$ arithmetic operations in $\mathbb{Z}_p$.

To solve the Vandermonde systems in Step 19 we implemented in C the quadratic algorithm of Zippel [38]. It does $O(s_{\rho,i}^2)$ arithmetic operations in $\mathbb{Z}_p$.

To call C code from Maple we use Maple's foreign function interface. This allows us to pass arrays of 32 bit or 64 bit integers between Maple and C.

3.1 Evaluating the factors

In Step 11 of Algorithm 1 we evaluate the factors $f_{\rho,j-1}(x_1, Y_k)$ for $1 \le k \le s$ where $Y_k = \beta_2^k, \ldots, \beta_{j-1}^k$. If we use Maple's `Eval(...) mod p` command, although coded in C, because Maple does each evaluation independently, it cannot take advantage of the geometric point sequence $Y_k = \beta_2^k, \ldots, \beta_{j-1}^k$. Suppose $f = \sum_{i=1}^{t} c_i x_1^{e_i} M_i(x_2, \ldots, x_{j-1})$ is one of the factors where the M_i are

monomials in $x_2, \ldots, x_{j-1}$. If we pre-compute the monomial evaluations $m_i = M_i(\beta_2, \ldots, \beta_{j-1})$, then we can exploit

$$M_i(\beta_2^k, \ldots, \beta_{j-1}^k) = M_i(\beta_2, \ldots, \beta_j)^k = m_i^k$$

to get a speedup as follows. Initializing $C = [c_1, \ldots, c_t]$ we compute

for $k = 1, 2, \ldots, s$ do
  1 $C \leftarrow [C_i \times m_i : 1 \le i \le t]$.
  2 $g \leftarrow \sum_{i=1}^{t} C_i x_1^{e_i}$. // $g = f(x_1, Y^k)$.

This algorithm does st multiplications in step 1 plus some additions in step 2 plus the work to compute the monomial evaluations m_i.

To compute $f(x_1, \beta_2^k, \ldots, \beta_{j-1}^k)$ using C code from Maple we first create the arrays C, m, e and g of size t, t, t, and $d_1 + 1$ respectively in Maple. We call a C program from Maple with the polynomial f and array β and output arrays C, m, and e. Then, for $k = 1, 2, \ldots, s$ we call two C programs from Maple. The first has inputs C, m and the prime p. It computes $C_i = C_i \times m_i \mod p$. The second has inputs C, e, g and p. It assembles the polynomial $\sum_{i=1}^{t} C_i x_1^{e_i}$ in g.

4 Benchmarks

We present two timing benchmarks. All timings were obtained on an Intel Xeon E5-2660 8 core CPU.

The first benchmark is shown in Tables 2 and 3. Table 2 shows the CPU time in seconds for computing the factors of $\det(T_n)$ where T_n is a symmetric Toeplitz matrix. We 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 this benchmark, we have implemented Bareiss' algorithm [2] to compute the determinant in $\mathbb{Z}_p$ for the black box in $O(n^2)$ arithmetic operations in $\mathbb{Z}_p$ rather than $O(n^3)$ by Gaussian elimination. For Maple we used Gentleman & Johnson's determinant algorithm [14] to compute $\det(T_n)$. Maple 2021 uses MTSHL [30] for factoring polynomials.

In Table 2, the first block of rows is the time for our new algorithm and the total number of probes to the black box it made. 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`, our implementation of the Gentleman-Johnson algorithm [14] and the time of Maple's `factor` command. The third block shows the time of determinant computation and factorization in Magma.

Table 3 is a summary of the breakdown of the 4 major steps of Algorithm 1 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. In the second block, BB and Interp2var are timings for probes to the black box and interpolation to get $a_j(x_1, Y_k, x_j)$ (step 10). Eval $f_{\rho,j-1}$ is the time for evaluating $f_{\rho,j-1}$ (step 11). BHL is the time for bivariate Hensel lifts (step 13). VSolve is the time for solving Vandermonde systems (step 19).

n	10	11	12	13	14	15	16
total time	5.790	13.430	50.855	154.441	722.310	1967.725	17,212.991
total 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 factor	1.91	3.48	23.11	57.75	509.82	7334.50	N/A
Maple total	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 2. CPU timings in seconds for computing $\det(T_n)$ using CMSHL black box factorization algorithm.

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 3. Breakdown of CPU timings in seconds for Hensel lifting the last variable x_n.

Our algorithm becomes faster at $n = 14$. Maple runs out of space (exceeds 72gigs) when trying to compute $\det(T_{16})$. Note that solving Vandermonde systems dominates the cost after $n = 13$. We could improve this by implementing the fast Vandermonde solver of Kaltofen and Yagati [19] which reduces Vandermonde solving to fast polynomial multiplication and division. In [7], Connolly's implementation of the fast Vandermonde solver from [19] first beats Zippel's $O(s^2)$ Vandermonde solver at size $s = 2024$.

Our 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 examples in the second benchmark. The second benchmark shows CPU times for factoring determinants which have four factors where the size of the factors is much smaller than the size of the determinant. The matrices are generated as follows. We first create two different $n \times n$ symmetric Toeplitz matrices T_1 and T_2 with multivariate polynomial entries. Then we create the $N \times N$ block diagonal matrix

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

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 4 and 5. In both tables, the first block of rows are the number of variables n and the matrix size N. In Table 4, the second block shows the total time for our algorithm, 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 our 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. The categories of Table 5 are the same as Table 3.

In this example, our algorithm is faster than Maple and Magma for all $n \geq 5$ and at $n = 8$, our algorithm is more than 330 times faster than Maple. Maple runs out of memory for computing $\det(A)$ at $n = 9$. The bottleneck is probes to the black box since evaluations of the polynomial entries are needed prior to each determinant computation in $\mathbb{Z}_p$.

n	5	6	7	8	9	10	11
$N = 2n$	10	12	14	16	18	20	22
total time	0.383	1.537	4.778	20.971	77.894	342.264	1334.654
total 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 factor	0.170	0.405	1.270	11.706	242.81	N/A	N/A
Maple total	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 4. CPU timings for computing $\det(A)$ using CMSHL black box algorithm.

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	0.029	0.039	0.116	0.514	1.580	6.362	21.586
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. Breakdown of CPU timings in seconds for Hensel lifting the last variable x_n.

5 Computing Determinants of Symbolic Matrices

In our first benchmark, factoring $\det T_n$, we first compared our new factorization algorithm with computing $\det T_n$ in its sparse representation using Maple's `LinearAlgebra:-Determinant` command then factoring it using Maple's polynomial factorization routine. Our initial timing results were impressive but we were fooled. The default algorithm Maple uses to compute $\det T_n$ is the Bareiss/ Edmonds fraction-free Gaussian elimination algorithm [1, 8]. For T_n a severe expression swell occurs. In this section we quantify this expression swell and consider two other symbolic determinant algorithms.

There are many algorithms for computing determinants of symbolic matrices. For our two benchmarks we tried (1) Maple's implementation of the Bareiss fraction-free algorithm [1], which was also discovered independently by Edmonds [8], (2) Gentleman & Johnson's method of minor expansion [14] and (3) the Berkowitz algorithm [4] as described by Law in [21]. For matrices over an integral domain D, these three algorithms do $O(n^3)$, $O(n2^n)$ and $O(n^4)$ arithmetic operations in D respectively. The Gentleman-Johnson algorithm and the Berkowitz

algorithm are division free algorithms. The Bareiss/Edmonds algorithm does exact divisions in D.

For the first benchmark, computing $\det(T_n)$ where T_n is the symmetric Toeplitz matrix with symbolic entries, the Gentleman-Johnson algorithm was the fastest. Maple uses this algorithm when we specify

`LinalgAlgebra:-Determinant(A,method=minor)`.

However, Maple ran out of space on T_{15}. To compute $\det(T_{15})$ we implemented the Gentleman-Johnson algorithm in Maple as follows. Let R_k denote the set of $k \times k$ submatrices of A with k rows chosen from 1 to n and the k rightmost columns of A. There are $\binom{n}{k}$ matrices in R_k. Let D_k be the determinants of those submatrices. Let us use $[r_1, \ldots, r_k][c_1, \ldots, c_k]$ to indicate the $k \times k$ submatrix of A with rows r_i and columns c_i. Then if $n = 4$ then

$R_1 = [1][4], [2][4], [3][4], [4][4],$
$R_2 = [1,2][3,4], [1,3][3,4], [1,4][3,4], [2,3][3,4], [2,4][3,4], [3,4][3,4],$
$R_3 = [1,2,3], [2,3,4], [1,2,4][2,3,4], [1,3,4][2,3,4], [2,3,4][2,3,4]$ and
$R_4 = [1,2,3,4][1,2,3,4] = A.$

The Gentleman-Johnson algorithm executes the following loop:

for $k = 2, 3, \ldots, n$ **do** compute the D_k using D_{k-1} **end for**

We modified the algorithm so that after computing D_k we explicitly discard D_{k-1} to save space in order to compute T_{15}. Maple cannot compute $\det(T_{16})$ as this polynomial will overflow Maple's POLY data structure forcing Maple to use the much slower and much less space efficient SUM-OF-PRODS data structure (see [25]).

Although the Bareiss/Edmonds fraction-free algorithm does only $O(n^3)$ polynomial operations, a significant intermediate expression swell occurs for the Toeplitz matrices T_n. We quantify this precisely. Ignoring pivoting, and initializing $A_{0,0} = 1$, the kth step of fraction-free Gaussian elimination computes $A_{i,j}$ as follows:

$$A_{i,j} = \frac{A_{k,k}A_{i,j} - A_{i,k}A_{k,j}}{A_{k-1,k-1}} \text{ for } k \le i \le n, k \le j \le n.$$

The division by $A_{k-1,k-1}$ is exact in our polynomial ring. Also, at the end of the elimination, $A_{k,k}$ is the determinant of the $k \times k$ principle minor of the original A, that is, $A_{k,k} = \det(T_k)$. Thus at the very last step of the algorithm we compute

$$\begin{aligned}\det(T_n) = A_{n,n} &= \frac{A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}}{A_{n-2,n-2}} \\ &= \frac{A_{n-1,n-1}A_{n,n} - A_{n,n-1}A_{n-1,n}}{\det(T_{n-2})}.\end{aligned}$$

Therefore the numerator polynomial is $\det(T_n)\det(T_{n-2})$. For $n = 10$ we find that $\#\det(T_{10}) = 23797$, $\#\det(T_8) = 1628$ and $\#\det(T_{10})\det(T_8) = 813638$, an expression swell of a factor of 34.

		Berkowitz		Gentleman		Bareiss		factor
n	$\#det$	real	cpu	real	cpu	real	cpu	real
10	5,318	1.48s	1.72s	1.45s	1.49s	1.43s	5.16s	0.282s
12	23,953	13.9s	22.7s	20.5s	21.3s	17.3s	112.4s	0.499s
14	231,714	6.85m	16.29m	9.14m	9.54m	15.0m	3.74h	2.676s
16	858,441	62.56m	3.11h	2.49h	2.61h	>72gb	–	21.9s
18	–	>72gb	–	>72gb	–	NA	–	–

Table 6. Maple timings for determinant algorithms for benchmark 2

For the second benchmark, we obtained the timing data in Table 6. The method that used the least CPU was the Gentleman-Johnson algorithm. The method that used the least real time was the Berkowitz algorithm. Interestingly, the Berkowitz algorithm uses significantly less real time than CPU time, the Bareiss/Edmonds algorithm uses much less real time than CPU time, but the Gentleman-Johnson algorithm does not. This is because when we multiply two polynomials $f \times g$ and both f and g have many terms, Maple is able to use it's parallel polynomial multiplication algorithm. In the Gentleman-Johnson algorithm, every polynomial multiplication $f \times g$, f is one of the original matrix entries which has few terms which limits parallel speedup.

6 Conclusion and Future Work

We have designed and implemented a new black box factorization algorithm that outputs the factors in sparse representation directly. It is highly space efficient and our determinant benchmarks show that the new algorithm is faster than Maple's current best determinant and factor algorithms. We were able to compute the factors of $\det(T_{16})$, where T_{16} is a symmetric Toeplitz matrix. This has never been computed before. To achieve these results we needed to implement several subroutines carefully in C. In the future, we want to try to parallelize our code using Cilk C, since each subroutine is parallelizable.

Acknowledgment

We would like to thank our colleague Garrett Paluck for his multi-factor bivariate Hensel lifting C code.

References

1. Bareiss, E.: Sylvester's Identity and multistep integer-preserving Gaussian elimination. *Math Comp.* **22**(103), 565–578 (1968)
2. Bareiss, E.: Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices. *Numerische Mathematik*, **13**(5), 404–424 (1969)

3. M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In Proceedings of STOC '88, pp. 301–309. ACM (1988)
4. Berkowitz, S.J.: On computing the determinant in small parallel time using a small number of processors. *Inf. Processing Letters* **18**(3), 147–150 (1984)
5. Bernardin, L.: On Bivariate Hensel Lifting and its Parallelization. In Proceedings of ISSAC '98, pp. 96–100. ACM (1998)
6. Chen, T., Monagan, M.: 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)
7. Connolly, K.: 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`
8. Edmonds, J.: Systems of Distinct Representatives and Linear Algebra *J. Research of the National Bureau of Standards*, **718**(4), 241–245 (1967)
9. Diaz, A., Kaltofen E.: FOXBOX: A system for manipulating symbolic objects in black box representation. In Proceedings of ISSAC '98, pp. 30–37. ACM (1998)
10. Von zur Gathen, J.: Irreducibility of multivariate polynomials. *Journal of Computer and System Sciences*, **31**(2), 225–264 (1985)
11. Von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press (2013)
12. Geddes, K.O., Czapor, S.R. and Labahn, G.: Algorithms for Computer Algebra. Kluwer Acad. Publ (1992)
13. David Hilbert. Über die Irreducibilität ganzer rational Functionen mit ganzzahigen Koeffizienten. *J. Renie Agewandte Mathematik*, **110**, 104–129 (1982)
14. Gentleman, W.M. and Johnson, S.C.: Analysis of algorithms, a case study: determinants of matrices with polynomial entries. *ACM Trans. on Math. Soft.* **2**(3), 232–241. ACM (1976)
15. Kaltofen, E.: Effective Hilbert irreducibility. *Information and Control*, **66**, 123–137 (1985)
16. Kaltofen, E.: Sparse Hensel lifting. In Proceedings of EUROCAL '85, LNCS **204**, 4–17. Springer (1985)
17. Khovanova, T., Scully, Z.: Efficient Calculation of Determinants of Symbolic Matrices with Many Variables, ArXiv: 1304.4691 (2013)
18. Kaltofen E., Trager, B.M.: 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)
19. Kaltofen, E., Yagati, L.: Improved sparse multivariate polynomial interpolation algorithms. In Proceedings of ISSAC '88, LNCS **358**, 467–474. Springer (1988)
20. Lang, S.: Fundamentals of Diophantine Geometry. Springer-Verlag New York (1983)
21. Law, M.: Computing characteristic polynomials of matrices of structured polynomials. Masters Thesis, Simon Fraser University, 2017.
22. Lecerf, G.: Improved dense multivariate polynomial factorization algorithms. *J. Symbolic Computation* **42**:477–494 (2007)
23. Lee, W.S.: Early termination strategies in sparse interpolation algorithms. Ph.D. Thesis (2001)
24. Monagan, M.: 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)

25. Monagan, M., Pearce, R.: The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Communications of Computer Algebra*, **48**(4), 166–186. ACM (2014)
26. Monagan, M., Tuncer, B.: Using Sparse Interpolation in Hensel Lifting. In Proceedings of CASC 2016, LNCS **9890**, 381–400. Springer (2016)
27. Monagan, M., Tuncer, B.: Factoring multivariate polynomials with many factors and huge coefficients. In Proceedings of CASC 2018, LNCS **11077**, 319–334. Springer (2018)
28. Monagan, M., Tuncer, B.: Sparse multivariate Hensel lifting: a high-performance design and implementation. In Proceedings of ICMS 2018, LNCS **10931**, 359–368. Springer (2018)
29. Monagan, M., Tuncer, B.: The complexity of sparse Hensel lifting and sparse polynomial factorization. *J. Symb. Cmpt.* **99**, 189–230. Elsevier (2020)
30. Monagan, M., Tuncer, B.: Polynomial factorization in Maple 2019. In Maple in Mathematics Education and Research. *Communications in Computer and Information Science*, **1125**, 341–345. Springer (2020)
31. Paluck, G., Monagan, M.: New bivariate Hensel lifting algorithm for n factors. *ACM Communications in Computer Algebra*, **53**(3), 142–145 (2019)
32. Roche, D.: What can (and can't) we do with sparse polynomials? In Proceedings of ISSAC 2018, pp. 25–30. ACM (2018)
33. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, **27**(4), 701–717 (1980)
34. Wang, P.S., Rothschild, L.P.: Factoring multivariate polynomials over the integers. *Math. Comp.* **29**, 935–950 (1975)
35. Yun, D.Y.Y.: The Hensel Lemma in algebraic manipulation. Ph.D. Thesis (1974)
36. Zippel, R.E.: Probabilistic algorithms for sparse polynomials. LNCS **72**, 216–226 (1979)
37. Zippel, R.E.: Newton's iteration and the sparse Hensel algorithm. In Proceedings of the ACM Symposium on Symbolic Algebraic Computation, pp. 68–72 (1981)
38. Zippel, R.E.: Interpolating polynomials from their values. *J. Symb. Cmpt.* **9**(3), 375–403 (1990)