Using Sparse Interpolation in Hensel Lifting

Michael Monagan and Baris Tuncer

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

Abstract. The standard approach to factor a multivariate polynomial in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ is to factor a univariate image in $\mathbb{Z}[x_1]$ then lift the factors of the image one variable at a time using Hensel lifting to recover the multivariate factors. At each step one must solve a multivariate polynomial Diophantine equation. For polynomials in many variables with many terms we find that solving these multivariate Diophantine equations dominates the factorization time. In this paper we explore the use of sparse interpolation methods, originally introduced by Zippel, to speed this up. We present experimental results in Maple showing that we are able to dramatically speed this up and thereby achieve a good improvement for multivariate polynomial factorization.

1 Introduction

Suppose that we seek to factor a multivariate polynomial $a \in R = \mathbb{Z}[x_1, \ldots, x_n]$ and $a = fg$ with f, g in R and $\gcd(f, g) = 1$. The multivariate Hensel lifting algorithm (MHL) developed by Yun [11] and improved by Wang [9, 10] uses a prime number p and an ideal $I = \langle x_2 - \alpha_2, \ldots, x_n - \alpha_n \rangle$ of $\mathbb{Z}_p[x_1, \ldots, x_n]$ where $\alpha_2, \alpha_3, \ldots, \alpha_n \in \mathbb{Z}_p$ is a random evaluation point chosen by the algorithm.

For a given polynomial $h \in R$, let us use the notation

$$h_j := h(x_1, \ldots, x_j, x_{j+1} = \alpha_{j+1}, \ldots, x_n = \alpha_n) \bmod p$$

so that $a_1 = a(x_1, \alpha_2, \ldots, \alpha_n) \bmod p$. The input to MHL is a, I, f_1, g_1 and p such that $a_1 = f_1 g_1$ and $\gcd(f_1, g_1) = 1$ in $\mathbb{Z}_p[x_1]$. The input factorization $a_1 = f_1 g_1$ is obtained by factoring $a(x_1, \alpha_2, \ldots, \alpha_n)$ over the integers. See [2].

Let d_j denote the total degree of a_j with respect to the variables $x_2, \ldots, x_j$ and $I_j = \langle x_2 - \alpha_2, \ldots, x_j - \alpha_j \rangle$ with $j \leq n$. Wang's MHL lifts the factorization $a_1 = f_1 g_1$ variable by variable to $a_j = f_j g_j \in \mathbb{Z}_p[x_1, \ldots, x_j]/I_j^{d_j+1}$. It turns out that $f_n \equiv f \bmod p$ and $g_n \equiv g \bmod p$. For sufficiently large p we recover the factorization of a over $\mathbb{Z}$.

We give a brief description of the j^{th} step of the MHL (assuming that the inputs are monic in the variable x_1, for simplicity) in algorithm 1 for $j > 1$. For details see [2]. There are two main sub-routines in the design of MHL. The first one is the leading coefficient correction algorithm. The most well-known is the Wang's heuristic leading coefficient algorithm [9] which works well in

practice and is the one Maple currently uses. There are other approaches by Kaltofen [3] and most recently by Lee [4]. In our implementation we use Wang's leading coefficient algorithm. The second main subroutine is the multivariate Diophantine problem (MDP). In MHL, for each j with $j \leq n$, Wang's design of MHL must solve many instances the MDP. In the Maple timings (see section 5), for most of the examples 90% of the time is spent solving MDPs.

In this paper we propose various approaches of sparse interpolation to solve MDP and present the results of our experiments. We will assume that a, f, g are monic in x_1 so as not to complicate the MHL algorithm with leading coefficient correction. In section 2 we define the MDP in detail. In section 3 we show that interpolation is an option to solve the MDP. If the factors to be computed are sparse then the solutions to the MDP are also sparse. We show in section 3.1 how to use Zippel's sparse interpolation to solve the MDP and we describe an improvement to the solution proposed in section 3.2. We have observed that often the evaluation cost is the most expensive part of these algorithms. In section 3.3 we will propose an improvement to the evaluation method used in the sparse interpolation process. Sparse Hensel Lifting (SHL) was first introduced by Zippel [14] and then improved by Kaltofen [3]. In section 4 we show that if we use Wang's leading coefficient correction then Kaltofen's SHL algorithm can be simplified, improved and implemented efficiently. Based on Lemma 1 in section 4 we will propose our SHL organization which is presented as algorithm 4. Finally in section 5 we will give some timing data to compare our factorization algorithms with Wang's algorithm, which is currently used by Maple.

2 The Multivariate Diophantine Problem (MDP)

Following the notation in section 1, let $u, w, c \in \mathbb{Z}_p[x_1, \ldots, x_j]$ in which u and w are monic polynomials with respect to the variable x_1 with $j \geqslant 1$ and let $I_j = \langle x_2 - \alpha_2, \ldots, x_j - \alpha_j \rangle$ be an ideal of $\mathbb{Z}_p[x_1, \ldots, x_j]$ with $\alpha_i \in \mathbb{Z}_p$. The MDP consists of finding multivariate polynomials $\sigma, \tau \in \mathbb{Z}_p[x_1, \ldots, x_j]$ that satisfy

$$\sigma u + \tau w = c \mod I_j^{d_j+1}$$

with $\deg_{x_1}(\sigma) < \deg_{x_1}(w)$ where d_j is the maximal total degree of σ and τ with respect to the variables $x_2, \ldots, x_j$ and it is given that

1. $\mathrm{GCD}(u, w) \mid c$ and
2. $\mathrm{GCD}\,(u \bmod I_j, w \bmod I_j) = 1 \,\mathrm{in}\, \mathbb{Z}_p[x_1]$.

It can be shown that the solution (σ, τ) exists and is unique provided the second condition is satisfied and that the solution is independent of the choice of the ideal I_j. For $j = 1$ the MDP is in $\mathbb{Z}_p[x_1]$ and can be solved with the extended Euclidean algorithm (see Chapter 2 of [2]).

It can be seen from algorithm 1 that at step j, there are at most $\max(\deg_{x_j}(f_j), \deg_{x_j}(g_j))$ calls to MDP. To solve the MDP for $j > 1$, Wang uses the same approach as for Hensel Lifting, that is, an ideal-adic approach (see [2]). In general,

Algorithm 1 j^{th} step of Multivariate Hensel Lifting for $j > 1$.

Input : $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ where a_j, f_{j-1}, g_{j-1} are monic in x_1 and $a_j(x_j = \alpha_j) = f_{j-1}g_{j-1}$.
Output : $f_j, g_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$ such that $a_j = f_j g_j$.

$\sigma_{j0} \leftarrow f_{j-1}$, $\tau_{j0} \leftarrow g_{j-1}$, $\sigma_j \leftarrow \sigma_{j0}$, $\tau_j \leftarrow \tau_{j0}$, $monomial \leftarrow 1$
$error \leftarrow a_j - f_{j-1}\, g_{j-1}$
for i from 1 to $\deg(a_j, x_j)$ **while** $error \neq 0$ **do**
    $monomial \leftarrow monomial \times (x_j - \alpha_j)$
    $c \leftarrow$ coefficient of $(x_j - \alpha_j)^i$ in the Taylor expansion of the $error$ about $x_j = \alpha_j$
    if $c \neq 0$ **then**
        Solve the MDP $\sigma_{ji}\tau_{j0} + \tau_{ji}\sigma_{j0} = c$ in $\mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ for σ_{ji} and τ_{ji}.
        $(\sigma_j, \tau_j) \leftarrow (\sigma_j + \sigma_{ji} \times monomial,\ \tau_j + \tau_{ji} \times monomial)$.
        $error \leftarrow a_j - \sigma_j\, \tau_j$.
    end if
end for
$f_j \leftarrow \sigma_j$ and $g_j \leftarrow \tau_j$

if $\alpha_k \neq 0$ for $j \leq k$, then an $\langle x_k - \alpha_k \rangle$-adic expansion of the solution is expensive to compute. Since even a sparse solution turns out to be dense in an $\langle x_k - \alpha_k \rangle$-adic expansion, the number of MDP's to be solved significantly increases and hence the time complexity of MHL becomes expensive. In the following sections we will present various approaches to solve the MDP.

3 Solution to the MDP via Interpolation

We consider whether we can interpolate $x_2, \ldots, x_j$ in σ. If $\beta \in \mathbb{Z}_p$ with $\beta \neq \alpha_j$, then we have

$$\sigma(x_j = \beta)u(x_j = \beta) + \tau(x_j = \beta)w(x_j = \beta) \;=\; c(x_j = \beta) \bmod I_{j-1}^{d_{j-1}+1}.$$

For $K_j = \langle x_2 - \alpha_2, \ldots, x_{j-1} - \alpha_{j-1}, x_j - \beta \rangle$ and $G_j = \mathrm{GCD}(u \bmod K_j,\ w \bmod K_j)$, we obtain a unique solution $\sigma(x_1, \ldots, x_{j-1}, \beta)$ iff $G_j = 1$. However it is possible that $G_j \neq 1$. Let $R = \mathrm{res}_{x_1}(u, w)$ be the Sylvester resultant of u and v taken in x_1. Since u, w are monic in x_1 one has

$$G_j \neq 1 \Longleftrightarrow \mathrm{res}_{x_1}(u \bmod K_j,\ w \bmod K_j) = 0 \Longleftrightarrow R \bmod K_j = 0.$$

Also $\deg(R) \leq \deg(u)\deg(w)$ [1]. Then by the Schwartz-Zippel Lemma [8, 13]

$$\mathrm{Prob}(G_j \neq 1) \leq \frac{\deg(u)\deg(w)}{p-1}.$$

If $\beta \neq \alpha_j$ is chosen at random and p is large, the probability that $G_j = 1$ is high so interpolation is thus an option to solve the MDP. If $G_j \neq 1$, we could choose another β but our implementation does not do this and simply returns FAIL. The bound above for $\mathrm{Prob}(G_j \neq 1)$ is a worst case bound. We note that in [6] we show that the average probability for $\mathrm{Prob}(G_j \neq 1) = 1/(p-1)$.

3.1 Solution to the MDP via Sparse Interpolation

Following the sparse interpolation idea of Zippel in [12], given a sub-solution $\sigma_j(x_1,\ldots,x_j=\alpha_j)$ for $\alpha_j \in \mathbb{Z}_p$ we use this information to create a sub-solution form σ_f and compute $\sigma_j(x_1,\ldots,x_j=\beta_j)$ for some other random $\beta_j \in \mathbb{Z}_p$ with high probability if p is big. Suppose the form of σ_j is

$$\sigma_f = \sum_{i=1}^{m} c_i(x_2,...,x_j)x_1^{n_i} \text{ where } c_i = \sum_{k=1}^{t_i} c_{ik}x_2^{\gamma_{2k}}\cdots x_j^{\gamma_{jk}} \text{ with } c_{ik} \in \mathbb{Z}_p\backslash\{0\}.$$

Let $t = \max_{i=1}^{m} t_i$ be the maximum number of terms in the coefficients of σ. In sparse interpolation we obtain each c_{ik} by solving m linear systems of size at most $t \times t$. As explained in [12], each linear system can be solved in $\mathcal{O}(t^2)$ arithmetic operations in $\mathbb{Z}_p$. We then interpolate x_j in σ_j from $\sigma_j(x_1,\ldots,x_{j-1},\beta_k)$ for $k = 0,\ldots,\deg_{x_j}(\sigma_j)$. Finally we compute $\tau_j = (c_j - \sigma_j u_j)/w_j$.

3.2 First Improvement

The approach introduced in the preceding section solves the interpolation problem based on projection down to $\mathbb{Z}_p[x_1]$. To reduce the cost we tried projecting down to $\mathbb{Z}_p[x_1,x_2]$ because this will likely reduce the number t of evaluation points needed. Let the total degree of σ in x_1, x_2 be bounded by d and let

$$\sigma_f = \sum_{i+k\le d} c_{ik}(x_3,...,x_j)x_1^i x_2^k \text{ where } c_{ik} = \sum_{l=0}^{s_{ik}} c_{ikl}x_3^{\gamma_{3l}}\cdots x_j^{\gamma_{jl}} \text{ with } c_{ikl} \in \mathbb{Z}_p\backslash\{0\}.$$

Let $s = \max s_{ik}$ be the maximum number of terms in the coefficients of σ_f. Here we solve $\mathcal{O}(d^2)$ linear systems of size at most $s \times s$. For $s < t$, the complexity of solving the linear systems decreases by a factor of $(t/s)^2$. We also save a factor t/s in the evaluation cost.

To solve the MDP in $\mathbb{Z}_p[x_1,x_2]$ we have implemented an efficient dense bivariate Diophantine solver (BDP) in C. The algorithm incrementally interpolates x_2 in both σ and τ from univariate images in $\mathbb{Z}_p[x_1]$. When σ and τ stabilize we test whether $\sigma(x_1,x_2)u(x_1,x_2) + \tau(x_1,x_2)w(x_1,x_2) = c(x_1,x_2)$ using sufficiently many evaluations to prove the correctness of the solution. The cost is $\mathcal{O}(d^3)$ arithmetic operations in $\mathbb{Z}_p$ where d bounds the total degree of c, u, w, σ and τ in x_1 and x_2. We do not compute τ using division because that would cost $\mathcal{O}(d^4)$ arithmetic operations. This bivariate MDP solving algorithm is presented as algorithm BSDiophant below.

3.3 The evaluation cost

In our experiments we found that the sparse interpolation approach we propose reduces the time spent solving MDPs but evaluation becomes the most time dominating part of the factoring algorithm.

Suppose $f = \sum_{i=1}^{s} c_i X_i Y_i$ where X_i is a monomial in x_1, x_2, Y_i is a monomial in $x_3,\ldots,x_n$, $0 \neq c_i \in \mathbb{Z}_p$ and we want to compute

$$f_j := f(x_1, x_2, x_3 = \alpha_3^j, \ldots, x_n = \alpha_n^j), \text{ for } j = 1,\ldots,t.$$

Algorithm 2 BSDiophant

Input A big prime p and $u, w, c \in \mathbb{Z}_p[x_1, x_2, \ldots, x_j]$.
Output $(\sigma, \tau) \in \mathbb{Z}_p[x_1, x_2, \ldots, x_j]$ such that $\sigma u + \tau w = c \in \mathbb{Z}_p[x_1, x_2, \ldots, x_j]$ or FAIL. It returns FAIL if condition 2 (see section 2) is not satisfied for the choice of any β in the algorithm. This is detected in subroutine BDP.

1: **if** $n = 2$ **then** call BDP to **return** $(\sigma, \tau) \in \mathbb{Z}_p[x_1, x_2]^2$ or FAIL **end if**.
2: Pick $\beta_1 \in \mathbb{Z}_p$ at random
3: $(u_{\beta_1}, w_{\beta_1}, c_{\beta_1}) \leftarrow (u(x_1, \ldots, x_j = \beta_1), w(x_1, \ldots, x_j = \beta_1), c(x_1, \ldots, x_j = \beta_1)$.
4: $(\sigma_1, \tau_1) \leftarrow$ **BSDiophant**$(u_{\beta_1}, w_{\beta_1}, c_{\beta_1}, p)$.
5: **if** $\sigma_1 =$ FAIL **then return** FAIL **end if**
6: $k \leftarrow 1$, $\sigma \leftarrow \sigma_1$, $q \leftarrow (x_j - \beta_1)$ and $\sigma_f \leftarrow$ skeleton of σ_1.
7: **repeat**
8: $\quad h \leftarrow \sigma$
9: $\quad$ Set $k \leftarrow k + 1$ and pick $\beta_k \in \mathbb{Z}_p$ at random distinct from $\beta_1, \ldots, \beta_{k-1}$
10: $\quad (u_{\beta_k}, w_{\beta_k}, c_{\beta_k}) \leftarrow (u(x_1, \ldots, x_j = \beta_k), w(x_1, \ldots, x_j = \beta_k), c(x_1, \ldots, x_j = \beta_k)$.
11: $\quad$ Solve the MDP $\sigma_k u_{\beta_k} + \tau_k w_{\beta_k} = c_{\beta_k}$ using sparse interpolation with σ_f.
12: $\quad$ **if** $\sigma_k =$ FAIL **then return** FAIL **end if**
13: $\quad$ Solve $\sigma = h \bmod q$ and $\sigma = \sigma_k \bmod (x_j - \beta_k)$ for $\sigma \in \mathbb{Z}_p[x_1, x_2, \ldots, x_j]$.
14: $\quad q \leftarrow q \cdot (x_j - \beta_k)$
15: **until** $\sigma = h$ and $w | (c - \sigma u)$
16: Set $\tau \leftarrow (c - \sigma u)/w$ and **return** (σ, τ).

To compute f_j efficiently, one way is to pre-compute the powers of α_i's in $(n-2)$ tables and then do the evaluation using tables. We implemented this first. Let $d_i = \deg(f, x_i)$ and $d = \max_{3 \le i \le n} d_i$. For a fixed j, computing the $n-2$ tables of powers of α_i^j's (i.e. $1, \alpha_i^j, \alpha_i^{2j}, \ldots, \alpha_i^{d_i j}$) costs $\le (n-2)d$ multiplications. To evaluate one term $c_i Y_i$ at $(\alpha_3^j, \ldots, \alpha_n^j)$ costs $n-2$ multiplications using the tables. Then the cost of evaluating f at $(\alpha_3^j, \ldots, \alpha_n^j)$ is $s(n-2)$ multiplications. Hence the total cost of t evaluations is bounded above by $C_T = s(n-2)t + (n-2)dt = t(n-2)(s+d)$ multiplications using tables.

However when we use sparse interpolation points of the form $(\alpha_3^j, \ldots, \alpha_n^j)$ for $j = 1, \ldots, t$ we can reduce the evaluation cost by a factor of $(n-2)$ by a simple organization. As an example suppose

$$f = x_1^{22} + 72x_1^3x_2^4x_4x_5 + 37x_1x_2^5x_3^2x_4 - 92x_1x_2^5x_5^2 + 6x_1x_2^3x_3x_4^2$$

and we want to compute $f_j := f(x_1, x_2, \alpha_3^j, \alpha_4^j, \alpha_5^j)$ for $1 \le j \le t$. Before combining and sorting, we write the terms of each f_j as

$$\begin{aligned} f_j &= x_1^{22} + 72\alpha_4^j\alpha_5^j x_1^3x_2^4 + 37(\alpha_3^j)^2\alpha_4^j x_1x_2^5 - 92(\alpha_5^j)^2 x_1x_2^5 + 6\alpha_3^j(\alpha_4^j)^2 x_1x_2^3 \\ &= x_1^{22} + 72(\alpha_4\alpha_5)^j x_1^3x_2^4 + 37(\alpha_3^2\alpha_4)^j x_1x_2^5 - 92(\alpha_5^2)^j x_1x_2^5 + 6(\alpha_3\alpha_4^2)^j x_1x_2^3. \end{aligned}$$

Now let

$$c^{(0)} := [1, 72, 37, -92, 6] \quad \text{and} \quad \theta := [1, \alpha_4\alpha_5, \alpha_3^2\alpha_4, \alpha_5^2, \alpha_3\alpha_4^2].$$

Then in a for loop $j = 1, \ldots, t$ we can update the coefficient array $c^{(0)}$ by the monomial array θ by defining $c_i^{(j)} = c_i^{(j-1)}\theta_i$ for $1 \le i \le s$ so that each iteration

computes the coefficient array

$$c^{(j)} = [1, 72(\alpha_4\alpha_5)^j, 37(\alpha_3^2\alpha_4)^j, -92(\alpha_5^2)^j, 6(\alpha_3\alpha_4^2)^j]$$

using $s = \#f$ multiplications in the coefficient field to obtain

$$f_j = x_1^{22} + 72(\alpha_4\alpha_5)^j x_1^3 x_2^4 + 37(\alpha_3^2\alpha_4)^j x_1 x_2^5 - 92(\alpha_5^2)^j x_1 x_2^5 + 6(\alpha_3\alpha_4^2)^j x_1 x_2^3.$$

Then sorting the monomials and combining terms we get

$$f_j = x_1^{22} + 72(\alpha_4\alpha_5)^j x_1^3 x_2^4 + (37(\alpha_3^2\alpha_4)^j - 92(\alpha_5^2)^j) x_1 x_2^5 + 6(\alpha_3\alpha_4^2)^j x_1 x_2^3.$$

Note that sorting is time consuming so it should be done once at the beginning.

With the organization described above one evaluates Y_i at $(\alpha_3, \ldots, \alpha_n)$ in $(n-3)$ multiplications using tables. The cost of $n-2$ tables of powers is $\leq (n-2)d$. Then at the first step the cost (of creating θ, the monomial array) is $\leq s(n-3)$. After that the cost of each evaluation is s multiplications. Hence the total cost is bounded above by $C_N = st + s(n-3) + (n-2)d$. Compared with $C_T = s(n-2)t + (n-2)dt$ the gain is a factor of $(n-2)$. Roman Pearce implemented this improved evaluation algorithm in C for us in such a way that from Maple, we can obtain the next evaluation using s multiplications.

4 Sparse Hensel Lifting

4.1 On Kaltofen's SHL

Factoring multivariate polynomials via Sparse Hensel Lifting (SHL) uses the same idea of the sparse interpolation [14]. Following the same notation introduced in section 1, at $(j-1)^{\text{th}}$ step we have $f_{j-1} = x_1^{df} + c_{j1}M_1 + \cdots + c_{jt_j}M_{t_j}$ where t_j is the number of non-zero terms that appear in f_{j-1}, M_k's are the distinct monomials in $x_1, \ldots, x_{j-1}$ and $c_{jk} \in \mathbb{Z}_p$ for $1 \leq k \leq t_j$. Then at the j^{th} step SHL assumes $f_j = x_1^{df} + \Lambda_{j1}M_1 + \cdots + \Lambda_{jt_j}M_{t_j}$ where for $1 \leq k \leq t_j$,

$$\Lambda_{jk} = c_{jk}^{(0)} + c_{jk}^{(1)}(x_j - \alpha_j) + c_{jk}^{(2)}(x_j - \alpha_j)^2 + \cdots + c_{jk}^{(d_{j_k})}(x_j - \alpha_j)^{d_{j_k}}$$

with $c_{jk}^{(0)} := c_{jk}$ and where $df = \deg_{x_1}(f)$, $d_{j_k} = \deg_{x_n}(\Lambda_{jk})$ with $c_{jk}^{(i)} \in \mathbb{Z}_p$ for $0 \leq i \leq d_{j_k}$. The assumption is the same for the factor g_{j-1}.

To recover f_j from f_{j-1} and g_j from g_{j-1}, during the j^{th} step of MHL (see algorithm 1 above) one starts with $\sigma_{j0} = f_{j-1}$, $\tau_{j0} = g_{j-1}$, then in a for loop starting from $i = 1$ and incrementing it while the error term and its i^{th} Taylor coefficient is non-zero, by solving MDP's $\sigma_{j0}\tau_{ji} + \tau_{j0}\sigma_{ji} = e_j^{(i)}$ for $1 \leq i \leq \max(\deg_{x_j}(f_j), \deg_{x_j}(g_j))$. After the loop terminates we have $f_j = \sum_{k=0}^{\deg_{x_j}(f_j)} \sigma_{jk}(x_j - \alpha_j)^k$. On the other hand if the assumption of SHL is true then we have also $f_j = x_1^{df} + (\sum_{i=0}^{d_j} c_{j1}^{(i)}(x_j - \alpha_j)^i)M_1 + \cdots + (\sum_{i=0}^{d_j} c_{jt_j}^{(i)}(x_j - \alpha_j)^i)M_{t_j}$ $= x_1^{df} + \sum_{i=0}^{d_j}(c_{j1}^{(i)}M_1 + \cdots + c_{jt_j}^{(i)}M_{t_j})(x_j - \alpha_j)^i$. Similarly for g_j.

Hence we see that if the assumption of SHL is true then the support of each σ_{jk} will be a subset of support of f_{j-1}. Therefore we can use f_{j-1} as the skeleton of the solution of each σ_{jk}. The same is true for τ_{jk}. Although it is not stated explicitly in [3], this is one of the underlying ideas of Kaltofen's SHL (KHL).

In a classical implementation of MHL, at the j^{th} step in the for loop (see algorithm 1) one gets the monic factors and then after the loop one applies leading coefficient correction. However in [3] leading coefficient correction is also done in the for loop. If we do leading coefficient correction after the for loop, Kaltofen's SHL idea reduces to solve the MDP by assuming for each $1 \leq i \leq d_j$, $\sigma_{ji} = u_1 M_1 + \cdots + u_{t_j} M_{t_j}$ and $\tau_{ji} = u_{t_j+1} N_1 + \cdots + u_{t_j+r_j} N_{r_j}$ for unknowns u_k and distinct monomials $M_1, \ldots, M_{t_j}$ and $N_1, \ldots, N_{r_j}$ in $x_1, \ldots, x_{j-1}$. Then by equating coefficients of the monomials appearing on the LHS and the RHS in the MDP equation one gets a linear system in the u_k's. By construction this system is homogeneous.

At the j^{th} step of MHL (see algorithm 1), throughout the loop σ_{j0} and τ_{j0} remain the same. So, if the SHL assumption is true the assumed solution structures of σ_{ji} and τ_{ji} will remain the same on the LHS and only the RHS of the MDP will change. Hence just before the loop it is sufficient to find $r_j + t_j$ linearly independent equations among $\mathcal{O}(r_j t_j)$ linear equations while keeping track of which monomials they correspond. We call this monomial set Mon, construct the corresponding matrix L, and compute L^{-1}. Then in the for loop, for each i, one simply has to compute the Taylor coefficient of $e_j^{(i)}$ of the error, extract the coefficients from it corresponding to each monomial in Mon, form the related vector v, and then compute $w = L^{-1}v$ to recover u_k's. This improvement makes the algorithm faster by a factor of $\deg(a_j, x_j)$.

We present the j^{th} step of KHL in algorithm 3. We give an example to show explicitly how it works in Appendix KHL.

Our organization of Kaltofen's approach needs no forward translation to $x_j \mapsto x_j + \alpha_j$ and not back translation $x_j + \alpha_j \mapsto x_j$, and also does not need to define the sets $E_{j-1}^{(i)}$ defined in [3]. This simplifies the algorithm.

Let $B = a_j(x_1, \ldots, x_j + \alpha_j, \alpha_{j+1}, \ldots, \alpha_n)$. Note that if we proceed in the way explained in [3] then for each i in the for loop we should compute

$$B - \left(f_j^{(i-1)} + (\sum_{k=1}^{t_j} u_k M_k) x_j^i \right) \left(g_j^{(i-1)} + (\sum_{k=1}^{r_j+t_j} u_k N_k) x_j^i \right) \tag{1}$$

where $f_j^{(i-1)} = \sum_{k=0}^{i-1} \sigma_{jk}(x_j - \alpha_j)^k$, $g_j^{(i-1)} = \sum_{k=0}^{i-1} \tau_{jk}(x_j - \alpha_j)^k$ and then by expanding (1) we need to form a linearly independent system by equating it with the error. Then we should apply back translation $x_j + \alpha_j \mapsto x_j$.

We have implemented our improved KHL in Maple. The most time consuming step is the step 7 of algorithm 3 where one has to find $r_j + t_j$ linearly independent equations out of $\mathcal{O}(r_j t_j)$ linear equations and invert the corresponding matrix. The most obvious way to get the linear system is to start with a set of one equation then add new equations to the set, one at at time, until the system has full rank $r_j + t_j$.

Algorithm 3 j^{th} step of improved Kaltofen's SHL for $j > 2$.

Input : $a_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ and $\alpha_j \in \mathbb{Z}_p$ where a_j, f_{j-1}, g_{j-1} are monic in x_1. Also, $a_j(x_1, \ldots, x_{j-1}, x_j = \alpha_j) = f_{j-1}g_{j-1}$.
Let $f_{j-1} = x_1^{df} + c_{j1}M_1 + \cdots + c_{jt_j}M_{t_j}$ and $g_{j-1} = x_1^{dg} + s_{j1}N_1 + \cdots + s_{jr_j}N_{r_j}$ where $M_1, \ldots, M_{t_j}$, and $N_1, \ldots, N_{r_j}$ are monomials in $x_1, \ldots, x_{j-1}$ and $df = \deg_{x_1} f$ and $dg = \deg_{x_1} g$.
Output : $f_j, g_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$ such that $a_j = f_j g_j$
or FAIL (No such factorization exists.)

1: $(\sigma_{j0}, \tau_{j0}) \leftarrow (f_{j-1}, g_{j-1})$.
2: $(\sigma_j, \tau_j) \leftarrow (\sigma_{j0}, \tau_{j0})$.
3: *monomial* $\leftarrow 1$.
4: Introduce unknowns $u_1, \ldots, u_{r_j+t_j}$ and $D \leftarrow \sigma_{j0}(u_1N_1 + \cdots + u_{r_j}N_{r_j}) + \tau_{j0}(u_{r_j+1}M_1 + \cdots + u_{t_j}M_{t_j})$
5: Expand D and collect the coefficients of the monomials in $x_1, \ldots, x_{j-1}$. Each coefficient is a homogeneous linear equation in u_k's.
6: Let S be the array of all these homogeneous equations and *Mon* be the array of monomials such that S_i is the coefficient of Mon_i in the expansion of D.
7: Find $i_1, \ldots, i_{r_j+t_j}$ such that $E = \{S_{i_1}, \ldots, S_{i_{r_j+t_j}}\}$ is a linearly independent set. Do this choosing equations of the form of $c\,u_k$ for some constant c first.
8: **if** no such E exists **then return** FAIL (SHL assumption is wrong) **end if**
9: Construct the $(r_j + t_j) \times (r_j + t_j)$ matrix L corresponding to the set E such that the unknown u_i corresponds to i^{th} column of L
10: Compute L^{-1}.
11: *error* $\leftarrow a_j - f_{j-1}\, g_{j-1}$
12: **for** i from 1 to $\deg(a_j, x_j)$ **while** *error* $\neq 0$ **do**
13: *monomial* $\leftarrow$ *monomial* $\times (x_j - \alpha_j)$
14: $c \leftarrow$ coefficient of $(x_j - \alpha_j)^i$ in the Taylor expansion of the *error* about $x_j = \alpha_j$
15: **if** $c \neq 0$ **then**
16: **for** k from 1 to $r_j + t_j$ **do**
17: $v_k \leftarrow$ the coefficient of Mon_{i_k} of the polynomial c
18: **end for**
19: $w \leftarrow L^{-1}v$
20: $\sigma_{ji} \leftarrow \sum_{k=1}^{t_j} w_k M_k$ and $\tau_{ji} \leftarrow \sum_{k=1}^{r_j} w_{k+t_j} N_k$.
21: $(\sigma_j, \tau_j) \leftarrow (\sigma_j + \sigma_{ji} \times monomial,\ \tau_j + \tau_{ji} \times monomial)$.
22: *error* $\leftarrow a_j - \sigma_j\tau_j$.
23: **end if**
24: **end for**
25: **if** *error* $\neq 0$ **then return** FAIL **else return** (σ_j, τ_j) **end if**

To implement this we use Maple's `RowReduce` function which performs in-place Gauss elimination on the input mod p Matrix L. This function is implemented in C and optimized. The time complexity is the time complexity of Gauss elimination $\mathcal{O}((r_j + t_j)^3)$ plus the time for the failed cases, which, according to our experiments, is not negligible. In our experiments we have observed that this approach is effective only when the factors are very sparse. According to our experiments in section 5.2, although our improved version of KHL is significantly faster than that described in [3], it is still slower than Wang's algorithm.

4.2 Our SHL organization

Before explaining our SHL organization we make the following observation:

Lemma 1. *Let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ and by* Support(f) *we denote the set of monomials present in f. Let α be a randomly chosen element in $\mathbb{Z}_p$ and $f = \sum_{i=0}^{d_n} b_i(x_1, \ldots, x_{n-1})(x_n - \alpha)^i$ be the $(x_n - \alpha)-$adic expansion of f, where $d_n = \deg_{x_n} f$. Then for a given j with $0 \leq j < d_n$,*

$$\text{Prob}(\text{Support}(b_{j+1}) \nsubseteq \text{Support}(b_j)) \leq |\text{Support}(b_{j+1})| \frac{d_n - j}{p - d_n + j + 1}.$$

Proof: For simplicity assume that $p > j$, otherwise we will need to introduce Hasse derivatives but the idea will be the same. We have

$$b_j(x_1, \ldots, x_{n-1}) = \frac{1}{j!} \frac{\partial}{\partial x_n^j} f(x_1, \ldots, x_{n-1}, x_n = \alpha).$$

If we write $f \in \mathbb{Z}_p[x_n][x_1, \ldots, x_{n-1}]$ as

$$f = c_1(x_n)M_1 + c_2(x_n)M_2 + \cdots + c_k(x_n)M_k$$

where $M_1, M_2, \ldots, M_k$ are the distinct monomials in $x_1, \ldots, x_{n-1}$ and we denote $\frac{\partial}{\partial x_n^j} c_i(x_n) = c_i^{(j)}(x_n)$ then

$$b_j = \frac{\partial}{\partial x_n^j} f(x_n = \alpha) = c_1^{(j)}(\alpha)M_1 + c_2^{(j)}(\alpha)M_2 + \cdots + c_k^{(j)}(\alpha)M_k.$$
$$b_{j+1} = \frac{\partial}{\partial x_n^{j+1}} f(x_n = \alpha) = c_1^{(j+1)}(\alpha)M_1 + c_2^{(j+1)}(\alpha)M_2 + \cdots + c_k^{(j+1)}(\alpha)M_k.$$

For a given $j > 0$, if $c_i^{(j+1)}(\alpha) \neq 0$, but $c_i^{(j)}(\alpha) = 0$ then $M_i \notin \text{Support}(b_j)$. We need to compute $\text{Prob}(c_i^{(j)}(\alpha) = 0 \,|\, c_i^{(j+1)}(\alpha) \neq 0)$. If A is the event that $c_i^{(j)}(\alpha) = 0$ and B is the event that $c_i^{(j+1)}(\alpha) = 0$ then

$$\text{Prob}(A \,|\, B^c) = \frac{\text{Prob}(A) - \text{Prob}(B)\text{Prob}(A \,|\, B)}{\text{Prob}(B^c)} \leq \frac{\text{Prob}(A)}{\text{Prob}(B^c)}.$$

By the Schwartz-Zippel Lemma[13, 8]

$$\frac{\text{Prob}(A)}{\text{Prob}(B^c)} \leq \frac{\deg_{x_n}(c_i^{(j)}(y))/p}{1 - (\deg_{x_n}(c_i^{(j+1)}(y))/p)} = \frac{(d_n - j)/p}{1 - (d_n - j - 1)/p} = \frac{d_n - j}{p - d_n + j + 1} \quad \square$$

Lemma 1 shows that for the sparse case, if p is big enough then the probability of $\text{Support}(b_{j+1}) \subseteq \text{Support}(b_j)$ is high.

Following the notation of Lemma 1 above, for a given $\alpha \in \mathbb{Z}_p$, let us call α unlucky, if $\text{Support}(b_{j+1}) \nsubseteq \text{Support}(b_j)$ for some $0 \leq j < d_n$. So, for a given f, if $c_i^{(j)}$ has a root but does not have a double root at $x_n = \alpha$, then α is unlucky for b_{j+1}, i.e. $\text{Support}(b_{j+1}) \nsubseteq \text{Support}(b_j)$: Consider the following example where $\text{Support}(b_{j+1}) \nsubseteq \text{Support}(b_j)$ for $j = 1, 2$.

$$f := (x_1^6 + x_1^5 + x_1^4)(x_2 - 1)^3 + (x_1^5 + x_1^4 + x_1^3)(x_2 - 1) + x_1^7 + 1 \in \mathbb{Z}_{509}[x_1, x_2].$$

But if we choose another point 301 and compute the $(x_2 - 301)-$adic expansion of $f = \sum_{i=0}^{3} b_i(x_1)(x_2 - 301)^i$ we have

$$\begin{aligned}
b_0 &= x_1^7 + 95x_1^6 + 395x_1^5 + 395x_1^4 + 300x_1^3 + 1 \\
b_1 &= 230x_1^6 + 231x_1^5 + 231x_1^4 + x_1^3 \\
b_2 &= 391x_1^6 + 391x_1^5 + 391x_1^4 \\
b_3 &= x_1^6 + x_1^5 + x_1^4
\end{aligned}$$

and we see that $\text{Support}(b_{j+1}) \subseteq \text{Support}(b_j)$ for $0 \leq j \leq 2$. In fact for this example $\alpha = 1, 209, -207$ are the only unlucky points as can be seen by considering $f \in \mathbb{Z}_{509}[x_2][x_1]$, that is,

$$\begin{aligned}
f = {} & x_1^7 + (x_2 - 1)^3 x_1^6 + (x_2 - 209)(x_2 - 1)(x_2 + 207)x_1^5 \\
& + (x_2 - 209)(x_2 - 1)x_1^4 + (x_2 - 1)x_1^3 + 1.
\end{aligned}$$

Note that these points are unlucky only for b_2. Before we give an upper bound for the number of unlucky points we consider the following example. Let $p = 1021$,

$$\begin{aligned}
f = {} & (x_2 - 841)(x_2 - 414)(x_2 - 15)(x_2 - 277)x_1^9 \\
& + (x_2 - 339)(x_2 - 761)(x_2 - 752)(x_2 - 345)x_1^7
\end{aligned}$$

and $f^{(i)} = \frac{\partial}{\partial x_2^i} f(x_1, x_2)$. Then

$$\begin{aligned}
f^{(1)} &= 4(x_2 - 384)(x_2 - 230)(x_2 - 291)x_1^9 + 4(x_2 - 441)(x_2 + 127)(x_2 + 453)x_1^7 \\
f^{(2)} &= 12(x_2 - 89)(x_2 - 174)x_1^9 + 12(x_2 - 473)(x_2 - 115)x_1^7 \\
f^{(3)} &= (24x_2 - 93)x_1^9 + (24x_2 + 91)x_1^7 \text{ and} \\
f^{(4)} &= 24x_1^9 + 24x_1^7.
\end{aligned}$$

So, the maximum number of unlucky points occurs if each $c_i^{(j)}$ splits for different points, hence $|\text{Support}(f)|\frac{d_n(d_n+1)}{p}$ is an upper bound for the probability of hitting an unlucky point. For a sparse polynomial with 1000 terms, $d_n = 20$, for $p = 2^{31} - 1$, this probability is 0.000097. This observation suggests that we use $\sigma_{i,j-1}$ (or $\tau_{i,j-1}$) as a form of the solution of σ_{ji} (or τ_{ij}).

Back to our discussion on SHL, based on the observation above the j^{th}step $(j > 1)$ of our SHL organization is summarized in algorithm 4. In Appendix SHL we give a concrete example to show how it works.

Algorithm 4 j^{th} step of Sparse Hensel Lifting for $j > 1$.

Input : $a_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$, $f_{j-1}, g_{j-1} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ and $\alpha_j \in \mathbb{Z}_p$ where a_j, f_{j-1}, g_{j-1} are monic in x_1. Also, $a_j(x_1, \ldots, x_{j-1}, x_j = \alpha_j) = f_{j-1}g_{j-1}$.
Output : $f_j, g_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$ such that $a_j = f_j g_j$ or FAIL (No such factorization exists.)

1: **if** $r_j > t_j$ **then** interchange f_{j-1} with g_{j-1} **end if**
2: $(\sigma_{j0}, \tau_{j0}) \leftarrow (f_{j-1}, g_{j-1})$.
3: $(\sigma_j, \tau_j) \leftarrow (\sigma_{j0}, \tau_{j0})$.
4: $monomial \leftarrow 1$.
5: $error \leftarrow a_j - f_{j-1}\, g_{j-1}$
6: **for** i from 1 to $\deg(a_j, x_j)$ **while** $error \neq 0$ **do**
7: $\quad monomial \leftarrow monomial \times (x_j - \alpha_j)$
8: $\quad c \leftarrow$ coefficient of $(x_j - \alpha_j)^i$ in the Taylor expansion of the $error$ about $x_j = \alpha_j$
9: $\quad$ **if** $c \neq 0$ **then**
10: $\quad\quad \sigma_g \leftarrow$ skeleton of $\tau_{j,i-1}$
11: $\quad\quad$ Solve the MDP $\sigma_{j0}\, \tau_{ji} + \tau_{j0}\, \sigma_{ji} = c$ for σ_{ji} and τ_{ji} in $\mathbb{Z}_p[x_1, \ldots, x_{j-1}]$
12: $\quad\quad$ using σ_g and our sparse interpolation from section 3.2.
13: $\quad\quad$ **if** (σ_{ji}, τ_{ji})=FAIL **then**
14: $\quad\quad\quad (\sigma_{ji}, \tau_{ji}) \leftarrow$ **BSDiophant**$(\sigma_{j0}, \tau_{j0}, c, p)$
15: $\quad\quad\quad$ **if** (σ_{ji}, τ_{ji})=FAIL **then** restart the factorization with a different ideal
16: $\quad\quad\quad$ **end if**
17: $\quad\quad$ **end if**
18: $\quad\quad (\sigma_j, \tau_j) \leftarrow (\sigma_j + \sigma_{ji} \times monomial,\ \tau_j + \tau_{ji} \times monomial)$.
19: $\quad\quad error \leftarrow a_j - \sigma_j \tau_j$.
20: $\quad$ **end if**
21: **end for**
22: **if** $error \neq 0$ **then return** FAIL (No such factorization exists)
23: **else return** (σ_j, τ_j)
24: **end if**

4.3 Some remarks on algorithm 4

Step 8 in the for loop computes the i^{th} Taylor coefficient of the error at $x_j = \alpha_j$. Maple used to compute this using the formula $c = g(x_j = \alpha_j)/i!$ where g is the i'th derivative of $error$ wrt x_j. Instead, Maple now uses the more direct formula $c = \sum_{k=i}^{d} \text{coeff}(error, x_j^k)\alpha_j^{k-i}\binom{k}{i}$ where $d = \deg_{x_j} error$ which is three times faster [7].

At step 10 algorithm 4 makes the assumption $\text{Support}(\tau_{ji}) \subseteq \text{Support}(\tau_{j,i-1})$ based on Lemma 1. Note that, if the minimum of the number of the terms of each factor of a_j is $t = \min(\#f_j, \#g_j)$, then at step 11 the probability of the failure of the assumption is $\leq t\frac{d_j - i}{p - d_j - (i-1)} \leq \frac{td_j}{p-2d_j}$ and its cost is the evaluation cost + cost of a system of linear equation solving which is bounded above by $\mathcal{O}(t^2)$. Another costly operation is the cost of multivariate division, $\sigma_{ji} = (c - \sigma_{j0}\tau_{ji})/\tau_{j0}$, which is hidden in sparse interpolation. If the algorithm fails to compute (σ_{ji}, τ_{ji}) at step 11 then it passes to a safe way at step 14.

Another expensive operation in the algorithm 4 is the error computation, $error \leftarrow a_j - \sigma_j\tau_j$, in the for loop. To decrease this cost, one of the ideas in [5] can be generalized to MHL. Let $\sigma_j = \sum_{s=0}^{\deg_{x_j}\sigma_j} \sigma_{j,s}(x_j-\alpha_j)^s$ and $\sigma_j^{(i)} = \sum_{s=0}^{i} \sigma_{j,s}(x_j-\alpha_j)^s$ (similarly for τ). One has

$$\begin{aligned} e_j^{(i+1)} &= a_j - \sigma_j^{(i)}\tau_j^{(i)} \\ &= a_j - (\sigma_j^{(i-1)} + \sigma_{j,i}(x_j-\alpha_j)^i)(\tau_j^{(i-1)} + \tau_{j,i}(x_j-\alpha_j)^i) \\ &= a_j - \sigma_j^{(i-1)}\tau_j^{(i-1)} - (\sigma_j^{(i-1)}\tau_{j,i} + \tau_j^{(i-1)}\sigma_{j,i})(x_j-\alpha_j)^i \\ &= e_j^{(i)} - U^{(i)}(x_j-\alpha_j)^i \end{aligned}$$

where $U^{(i)} := (\sigma_j^{(i-1)}\tau_{j,i} + \tau_j^{(i-1)}\sigma_{j,i})$. Hence in the for loop we have the relation $e_j^{(i+1)} = e_j^{(i)} - (x_j-\alpha_j)^i U^{(i)}$ for a correction term $U^{(i)} \in \mathcal{O}((x_j-\alpha_j)^{i-1})$. Also for $i \geqslant 0$ it is known that $(x_j-\alpha_j)^i$ divides $e_j^{(i)}$. So if we define $c_j^{(i)} := e_j^{(i)}/(x_j-\alpha_j)^i$ then $c_j^{(i)}$ can be computed efficiently using

$$c_j^{(i+1)} = (c_j^{(i)} - U^{(i)})/(x_j-\alpha_j).$$

Hence we may compute $c_j^{(i)}$ for $i = 1, 2, \ldots$ until it becomes zero instead of computing $e_j^{(i)}$. According to our experiments, this observation decreases the cost when the number of factors is 2. For more than 2 factors, the generalization of it does not bring a significant advantage. So, in our implementations we only use this update formula when the number of factors is 2.

Also note that, in our SHL organization (algorithm 4), we use only one of the SHL assumptions and eliminate the recursive step in MHL to compute the skeleton of the solution. In Kaltofen's approach one cannot focus on some subset of the u_k's as we do, since the system of equations are coupled.

5 Some Timing Data

To compare the result of our ideas with Wang's, first we factored the determinants of Toeplitz and Cyclic matrices of different sizes as concrete examples. Note that the factors in these concrete examples are not sparse. Our results are presented in section 5.1. Then we created sparse random polynomials A, B using

$$x_1^d + \texttt{randpoly}([x_2, .., x_n], \texttt{degree} = d, \texttt{terms} = t)$$

in Maple and computed $C = AB \in R$. Note that we chose monic factors in x_1 so as not to complicate the algorithm with leading coefficient correction and to have a fair comparison with Maple's factorization algorithm. We used $p = 2^{31}-1$ and two ideal types for factoring C: ideal type 1: $I = \langle x_2-0, x_3-0, \cdots, x_n-0\rangle$ and ideal type 2: $I = \langle x_2-\alpha_1, x_3-\alpha_2, \cdots, x_n-\alpha_n\rangle$ where the α_i's in practice are small. However for sparse Hensel liftings, as explained in section 4, it is

important that α_i's should be chosen from a large interval. For these we chose α_i's randomly from $\mathbb{Z}_q - \{0\}$ with $q = 65521$. Our results are presented in section 5.2. In section 5.2 we also included the ideal type 1 case since according to our experiments it is the only case where Wang's algorithm is faster. This is because a sparse polynomial remains sparse for the ideal type 1 and hence the number of MDP's to be solved significantly decreases and the evaluation cost of sparse interpolation becomes dominant which is not the case for Wang's algorithm for the ideal type 1 case. However it is not always possible to use ideal type 1. For example, ideal type 1 cannot be used to factor Cyclic or Toeplitz determinants.

In the tables below all timings are in CPU seconds and are for the Hensel liftings part of the polynomial factorization. They were obtained on an Intel Core i5–4670 CPU running at 3.40GHz.

tW is the time for Wang's algorithm which Maple currently uses (see[2]),
tUW is the time for Wang's algorithm with the improved Hensel,
tS is the time for Zippel's sparse interpolation from section 3.1,
tBS is the time for the improved sparse interpolation from section 3.2,
tKHL is the time for the Kaltofen's sparse Hensel lifting from section 4.1,
tNBS is the time for the sparse Hensel lifting from section 4.2,
tX(tY) means factoring time tX with tY seconds spent solving MDPs.

5.1 Factoring the determinants of Cyclic and Toeplitz matrices

Let C_n denote the $n \times n$ cyclic matrix and let T_n denote the $n \times n$ symmetric Toeplitz matrix below.

$$C_n = \begin{pmatrix} x_1 & x_2 & \dots & x_{n-1} & x_n \\ x_n & x_1 & \dots & x_{n-2} & x_{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_3 & x_4 & \dots & x_1 & x_2 \\ x_2 & x_3 & \dots & x_n & x_1 \end{pmatrix} \quad \text{and} \quad T_n = \begin{pmatrix} x_1 & x_2 & \cdots & x_{n-1} & x_n \\ x_2 & x_1 & \cdots & x_{n-2} & x_{n-1} \\ & \ddots & \ddots & \ddots & \\ x_{n-1} & x_{n-2} & \cdots & x_1 & x_2 \\ x_n & x_{n-1} & \cdots & x_2 & x_1 \end{pmatrix}$$

The determinants of C_n and T_n are polynomials in n variables $x_1, x_2, \dots, x_n$ which factor. For $n > 1$ $\det(T_n)$ has 2 factors and $x_1 + x_2 + \cdots + x_n$ is a factor of C_n. Table 1 presents timings for Hensel liftings in CPU seconds to factor $\det C_n$. For $n = 6, 10, 12$ the number of factors is 3,4 and 6 respectively. For $n = 5, 7, 11, 13$ the number of factors is 2. We didn't implement KHL to factor more than 2 factors. This is why we didn't include the timing for KHL for the case $n = 6$. As can be seen from the data below KHL is not effective for $n \geq 7$. Table 2 presents timings for Hensel liftings in CPU seconds to factor $\det T_n$.

5.2 Random data

Table 3 below presents timings for the random data where ideal type 1 is used. For the ideal type 1 case SHL is not used, since the zero evaluation probability is high for the sparse case. Table 4 below presents timings for the random data where ideal type 2 is used. As can be seen KHL is effective only when the factors have 100 terms or less.

Table 1. Timings (CPU seconds) for factoring determinants of $n \times n$ cyclic matrices.

n	tW	tUW	$tKHL$	tS	tBS	$tNBS$
5	0.004 (0.003)	0.014 (0.013)	0.07 (0.068)	0.014 (0.003)	0.015 (0.012)	0.014 (0.012)
7	0.057 (0.054)	0.054 (0.04)	1157.(1157.)	0.018 (0.006)	0.019 (0.014)	0.017 (0.014)
10	0.912 (0.666)	-	-	1.049 (0.823)	0.775 (0.549)	0.434 (0.179)
11	9.437 (8.785)	8.413 (8.107)	∞	0.503 (0.23)	0.505 (0.226)	0.354 (0.071)
12	42.64 (38.38)	-	-	7.705 (4.35)	7.288 (3.913)	4.372 (1.047)
13	258.5 (208.9)	256.5 (208.9)	∞	20.40 (8.936)	20.05 (8.408)	13.78 (1.697)

Table 2. Timings for factoring determinants of $n \times n$ symmetric Toeplitz matrices.

n	tW	tUW	$tKHL$	tS	tBS	$tNBS$
5	0.003 (0.002)	0.014 (0.001)	0.02 (0.018)	0.014 (0.014)	0.017 (0.017)	0.015 (0.012)
6	0.016 (0.013)	0.016 (0.005)	0.308 (0.306)	0.04 (0.026)	0.042 (0.031)	0.021 (0.008)
7	0.025 (0.012)	0.044 (0.029)	1157.5(1157.5)	0.031 (0.019)	0.032 (0.019)	0.045 (0.03)
8	0.057 (0.044)	0.072 (0.052)	119.88(119.86)	0.103 (0.086)	0.096 (0.087)	0.059 (0.026)
9	0.167 (0.126)	0.151 (0.123)	486.45(486.41)	0.279 (0.258)	0.194 (0.168)	0.088 (0.06)
10	0.654 (0.461)	0.629 (0.496)	25021.(25021.)	1.389 (1.245)	0.675 (0.531)	0.366 (0.222)
11	2.699 (2.06)	2.538 (2.11)	∞	7.612 (7.109)	2.677 (1.751)	1.133 (0.589)
12	25.93 (18.68)	23.07 (17.95)	∞	69.91 (65.8)	22.08 (15.72)	13.86 (7.579)
13	48.59 (37.43)	47.01 (37.73)	∞	508.3 (495.8)	48.86 (36.11)	32.81 (20.36)

6 Conclusion

We have shown that solving the multivariate polynomial diophantine equations in sparse Hensel lifting algorithm can be improved by using sparse interpolation. This leads to an overall improvement in multivariate polynomial factorization. Our experiments show that the improvement is practical.

Appendix KHL

Suppose we seek to factor $a = fg$ where $f = {x_1}^5 + 3\,{x_1}^2 x_2 {x_3}^2 - 7\,{x_1}^4 - 4\,x_1 x_3 + 1$ and $g = {x_1}^5 + {x_1}^2 x_2 x_3 - 7\,{x_3}^4 - 6$. Let $\alpha_3 = 2$ and $p = 2^{31} - 1$. Before lifting we have a and

$$f^{(0)} := f(x_3 = 2) = {x_1}^5 - 7\,{x_1}^4 + 12\,{x_1}^2 x_2 - 8\,x_1 + 1$$
$$g^{(0)} := g(x_3 = 2) = {x_1}^5 + 2\,{x_1}^2 x_2 - 118.$$

If the assumption of SHL is true then we assume that $f = \sum_{i=0}^{\deg_{x_3} f} f_i (x_3 - 2)^i$ and $g = \sum_{i=0}^{\deg_{x_3} g} g_i (x_3 - 2)^i$ where each f_i and g_i are in the form

$$f_i = c_1 {x_1}^4 + c_2 {x_1}^2 x_2 + c_3 x_1 + c_4 \quad \text{and} \quad g_i = c_5 {x_1}^2 x_2 + c_6$$

for some unknowns $C = \{c_1, c_2, c_3, c_4, c_5, c_6\}$. First we construct

$$D = \left({x_1}^5 - 7\,{x_1}^4 + 12\,{x_1}^2 x_2 - 8\,x_1 + 1\right)\left(c_5 {x_1}^2 x_2 + c_6\right) + \left({x_1}^5 + 2\,{x_1}^2 x_2 - 118\right)\left(c_1 {x_1}^4 + c_2 {x_1}^2 x_2 + c_3 x_1 + c_4\right).$$

Table 3. The timing table for random data with ideal type 1

$n/d/t$	tW	tUW	tS	tBS
3/35/100	0.11 (0.06)	0.10 (0.06)	0.17 (0.13)	0.07 (0.03)
3/35/500	0.39 (0.16)	0.44 (0.17)	0.60 (0.36)	0.31 (0.08)
5/35/100	0.183 (0.15)	0.18 (0.15)	0.46 (0.43)	0.72 (0.69)
5/35/500	1.42 (0.53)	2.61 (0.51)	5.25 (2.92)	5.05 (2.68)
7/35/100	0.18 (0.16)	0.18 (0.15)	0.79 (0.76)	1.05 (1.02)
7/35/500	1.48 (0.71)	2.36 (0.65)	12.44 (10.43)	7.77 (5.61)

Table 4. The timing table for random data with ideal type 2

$n/d/t$	tW	tUW	$tKHL$	tS	tBS	$tNBS$
3/35/100	2.87 (1.88)	2.14 (1.88)	0.401 (0.046)	0.65 (0.38)	0.38 (0.08)	0.32 (0.04)
3/35/500	5.77 (3.69)	4.30 (3.57)	1.957 (0.057)	1.36 (0.61)	0.90 (0.14)	0.81 (0.05)
5/35/100	88.10 (86.28)	86.45 (85.64)	3.337 (2.551)	6.12 (5.21)	5.04 (4.11)	1.16 (0.36)
5/35/500	472.1 (402.5)	392.2 (370.7)	3732. (3717.)	67.57 (45.98)	48.1 (25.5)	26.0 (4.86)
6/35/100	309.1 (306.3)	323.8 (322.6)	4.383 (3.409)	12.53 (11.42)	9.29 (7.11)	1.49 (0.46)
7/35/100	800.0 (797.0)	829.7 (828.5)	10.22 (9.134)	16.82 (15.15)	10.8 (9.77)	1.58 (0.59)

Expanding D we see the system of homogeneous linear equations as coefficients

$$\begin{aligned} D = {} & c_1 {x_1}^9 + (c_2 + c_5)\, {x_1}^7 x_2 + (2\,c_1 - 7\,c_5)\, {x_1}^6 x_2 + c_3 {x_1}^6 \\ & + (c_4 + c_6)\, {x_1}^5 + (2\,c_2 + 12\,c_5)\, {x_1}^4 {x_2}^2 \\ & + (-118\,c_1 - 7\,c_6)\, {x_1}^4 + (2\,c_3 - 8\,c_5)\, {x_1}^3 x_2 + (-118\,c_2 + 2\,c_4 + c_5 + 12\,c_6)\, {x_1}^2 x_2 \\ & + (-118\,c_3 - 8\,c_6)\, x_1 - 118\,c_4 + c_6 \end{aligned}$$

We need 6 linearly independent equations from these. First we check whether there are single equations. In this example we see that c_1 and c_3 corresponding to monomials ${x_1}^9, {x_1}^6$. Then we go over the equations one by one to get a rank 6 system. In this example we see that equations corresponding to the set $Mon = \{{x_1}^9, {x_1}^6, {x_1}^7 x_2, {x_1}^6 x_2, {x_1}^5, {x_1}^4\}$ are linearly independent. We obtain

$$L = \begin{bmatrix} 1\,0\,0\,0 & 0 & 0 \\ 0\,0\,1\,0 & 0 & 0 \\ 0\,1\,0\,0 & 1 & 0 \\ 2\,0\,0\,0 & -7 & 0 \\ 0\,0\,0\,1 & 0 & 1 \\ -118\,0\,0\,0 & 0 & -7 \end{bmatrix}$$

and compute L^{-1}. In the following $e_3^{(k)}$ denotes the coefficient of $(x_3 - 2)^k$ in the Taylor expansion of the *error* about $x_3 = 2$. Let also $f_0 := f^{(0)}, g_0 := g^{(0)}, f^{(k)} := \sum_{i=0}^{k} f_i (x_3 - 2)^i, g^{(k)} := \sum_{i=0}^{k} g_i (x_3 - 2)^i$.

In algorithm 3 v is the vector constructed by extracting the coefficients of $e_3^{(k)}$ corresponding to monomials in $Mon = \{{x_1}^9, {x_1}^6, {x_1}^7 x_2, {x_1}^6 x_2, {x_1}^5, {x_1}^4\}$ and $w = L^{-1} v \bmod p$. Now for the loop,

Step $i = 1$: error $= a - f^{(0)}g^{(0)}$

$$\begin{aligned}
e_3^{(1)} &= 13\,{x_1}^7x_2 - 7\,{x_1}^6x_2 - 4\,{x_1}^6 + 36\,{x_1}^4{x_2}^2 - 224\,{x_1}^5 \\
&\quad +1568\,{x_1}^4 - 16\,{x_1}^3x_2 - 4103\,{x_1}^2x_2 + 2264\,x_1 - 224 \\
v &= \begin{bmatrix} 0 & -4 & 13 & -7 & -224 & 1568 \end{bmatrix} \\
w &= L^{-1}v = \begin{bmatrix} 0 & 12 & -4 & 0 & 1 & -224 \end{bmatrix} \\
f^{(1)} &= f^{(0)} + \left(12\,{x_1}^2x_2 - 4\,x_1\right)(x_3 - 2) \\
&= {x_1}^5 - 7\,{x_1}^4 + 12\,{x_1}^2x_2x_3 - 12\,{x_1}^2x_2 - 4\,x_1x_3 + 1 \\
g^{(1)} &= g^{(0)} + \left({x_1}^2x_2 - 224\right)(x_3 - 2) = {x_1}^5 + {x_1}^2x_2x_3 - 224\,x_3 + 330
\end{aligned}$$

Step $i = 2$: error $= a - f^{(1)}g^{(1)}$

$$\begin{aligned}
e_3^{(2)} &= 3\,{x_1}^7x_2 + 6\,{x_1}^4{x_2}^2 - 168\,{x_1}^5 + 1176\,{x_1}^4 - 2370\,{x_1}^2x_2 + 1344\,x_1 - 168 \\
v &= \begin{bmatrix} 0 & 0 & 3 & 0 & -168 & 1176 \end{bmatrix} \\
w &= L^{-1}v = \begin{bmatrix} 0 & 3 & 0 & 0 & 0 & -168 \end{bmatrix} \\
f^{(2)} &= f^{(1)} + 3\,{x_1}^2x_2(x_3 - 2)^2 = {x_1}^5 + 3\,{x_1}^2x_2{x_3}^2 - 7\,{x_1}^4 - 4\,x_1x_3 + 1 \\
g^{(2)} &= g^{(1)} - 168(x_3 - 2)^2 = {x_1}^5 + {x_1}^2x_2x_3 - 168\,{x_3}^2 + 448\,x_3 - 342
\end{aligned}$$

Step $i = 3$: error $= a - f^{(2)}g^{(2)}$

$$\begin{aligned}
e_3^{(3)} &= -56\,{x_1}^5 + 392\,{x_1}^4 - 672\,{x_1}^2x_2 + 448\,x_1 - 56 \\
v &= \begin{bmatrix} 0 & 0 & 0 & 0 & -56 & 392 \end{bmatrix} \\
w &= L^{-1}v = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -56 \end{bmatrix} \\
f^{(3)} &= f^{(2)} + 0 = {x_1}^5 + 3\,{x_1}^2x_2{x_3}^2 - 7\,{x_1}^4 - 4\,x_1x_3 + 1 \\
g^{(3)} &= g^{(2)} - 56\,(x_3 - 2)^3 = {x_1}^5 + {x_1}^2x_2x_3 - 56\,{x_3}^3 + 168\,{x_3}^2 - 224\,x_3 + 106
\end{aligned}$$

At the end of the 3rd iteration we have recovered f actually and so we could obtain $g = a/f$ via trial division and terminate. But let's go further.

Step $i = 4$: error $= a - f^{(3)}g^{(3)}$

$$\begin{aligned}
e_3^{(4)} &= -7\,{x_1}^5 + 49\,{x_1}^4 - 84\,{x_1}^2x_2 + 56\,x_1 - 7 \\
v &= \begin{bmatrix} 0 & 0 & 0 & 0 & -7 & 49 \end{bmatrix} \\
w &= L^{-1}v = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -7 \end{bmatrix} \\
f^{(4)} &= f^{(3)} + 0 = {x_1}^5 + 3\,{x_1}^2x_2{x_3}^2 - 7\,{x_1}^4 - 4\,x_1x_3 + 1 \\
g^{(4)} &= g^{(3)} - 7\,(x_3 - 2)^4 = {x_1}^5 + {x_1}^2x_2x_3 - 7\,{x_3}^4 - 6
\end{aligned}$$

for $i = 5$, $error = a - f^{(4)}g^{(4)} = 0$ and we have the factors!

Appendix SHL

We give an example of our SHL. Suppose we seek to factor $a = fg$ where

$$f = {x_1}^8 + 2\,x_1{x_2}^2{x_4}^3x_5 + 4\,x_1{x_2}^2{x_3}^3 + 3\,x_1{x_2}^2x_4{x_5}^2 + {x_2}^2x_3x_4 - 5$$
$$g = {x_1}^8 + 3\,{x_1}^2x_2x_3{x_4}^2x_5 + 5\,{x_1}^2x_2{x_3}^2x_4 - 3\,{x_4}^2{x_5}^2 + 4\,x_5$$

Let $\alpha_3 = 1, p = 2^{31} - 1$. Before lifting x_5 we have

$$f^{(0)} := f(x_5 = 1) = {x_1}^8 + 4\,x_1{x_2}^2{x_3}^3 + 2\,x_1{x_2}^2{x_4}^3 + 3\,x_1{x_2}^2x_4 + {x_2}^2x_3x_4 - 5$$
$$g^{(0)} := g(x_5 = 1) = {x_1}^8 + 5\,{x_1}^2x_2x_3{x_4}^2 + 3\,{x_1}^2x_2x_3{x_4}^2 - 3\,{x_4}^2 + 4$$

satisfying $a(x_5 = \alpha_5) = f^{(0)}g^{(0)}$. If the SHL assumption is true then at the first step we assume $f = \sum_{i=0}^{\deg_{x_5} f} f_i(x_5 - 1)^i$ and $g = \sum_{i=0}^{\deg_{x_5} g} g_i(x_5 - 1)^i$ where f_1 and g_1 are in the form

$$f_1 = \left(c_1{x_3}^3 + c_2{x_4}^3 + c_3x_4\right)x_1{x_2}^2 + c_4{x_2}^2x_3x_4 + c_5$$
$$g_1 = \left(c_6{x_3}^2x_4 + c_7x_3{x_4}^2\right){x_1}^2x_2 + c_8{x_4}^2 + c_9$$

for some unknowns $C = \{c_1, \ldots, c_9\}$. In the following $e_5^{(k)}$ denotes the coefficient of $(x_5-1)^k$ in the Taylor expansion of the *error* about $x_5 = 1$. Let also $f_0 := f^{(0)}$, $g_0 := g^{(0)}$, $f^{(k)} := \sum_{i=0}^{k} f_i(x_5 - 1)^i$ and $g^{(k)} := \sum_{i=0}^{k} g_i(x_5 - 1)^i$.
We start by computing the first error term $e_5^{(1)} = a - f^{(0)}g^{(0)}$. We obtain

$$\begin{aligned} e_5^{(1)} = {} & 3\,{x_1}^{10}x_2x_3{x_4}^2 + 2\,{x_1}^9{x_2}^2{x_4}^3 + 6\,{x_1}^9{x_2}^2x_4 + 12\,{x_1}^3{x_2}^3{x_3}^4{x_4}^2 + 10\,{x_1}^3{x_2}^3{x_3}^2{x_4}^4 \\ & + 12\,{x_1}^3{x_2}^3x_3{x_4}^5 - 6\,{x_1}^8{x_4}^2 + 30\,{x_1}^3{x_2}^3{x_3}^2{x_4}^2 + 27\,{x_1}^3{x_2}^3x_3{x_4}^3 + 3\,{x_1}^2{x_2}^3{x_3}^2{x_4}^3 \\ & + 4\,{x_1}^8 - 24\,x_1{x_2}^2{x_3}^3{x_4}^2 - 18\,x_1{x_2}^2{x_4}^5 - 15\,{x_1}^2x_2x_3{x_4}^2 + 16\,x_1{x_2}^2{x_3}^3 \\ & - 20\,x_1{x_2}^2{x_4}^3 - 6\,{x_2}^2x_3{x_4}^3 + 36\,x_1{x_2}^2x_4 + 4\,{x_2}^2x_3x_4 + 30\,{x_4}^2 - 20 \end{aligned}$$

The MDP to be solved is:

$$\begin{aligned} D := {} & f_0\left(\left(c_6{x_3}^2x_4 + c_7x_3{x_4}^2\right){x_1}^2x_2 + c_8{x_4}^2 + c_9\right) \\ & + g_0\left(\left(c_1{x_3}^3 + c_2{x_4}^3 + c_3x_4\right)x_1{x_2}^2 + c_4{x_2}^2x_3x_4 + c_5\right) = e_5^{(1)}. \end{aligned}$$

Our aim is first to get $\left(\left(c_6{x_3}^2x_4 + c_7x_3{x_4}^2\right){x_1}^2x_2 + c_8{x_4}^2 + c_9\right)$ since it will create a smaller matrix. For sparse interpolation we need 2 evaluations only: we choose $[x_3 = 2, x_4 = 3]$ and $[x_3 = 2^2, x_4 = 3^2]$ and compute $D([x_3 = 2, x_4 = 3])$:

$$\begin{aligned} & \left({x_1}^8 + 95\,x_1{x_2}^2 + 6\,{x_2}^2 - 5\right)\left(\left(12\,c_6 + 18\,c_7\right){x_1}^2x_2 + 9\,c_8 + c_9\right) \\ & \quad + \left({x_1}^8 + 114\,{x_1}^2x_2 - 23\right)\left(\left(8\,c_1 + 27\,c_2 + 3\,c_3\right)x_1{x_2}^2 + 6\,c_4{x_2}^2 + c_5\right) \\ & = 54\,{x_1}^{10}x_2 + 72\,{x_1}^9{x_2}^2 - 50\,{x_1}^8 + 13338\,{x_1}^3{x_2}^3 + 324\,{x_1}^2{x_2}^3 - 270\,{x_1}^2x_2 \\ & \qquad - 6406\,x_1{x_2}^2 - 300\,{x_2}^2 + 250 \end{aligned}$$

and $D([x_3 = 4, x_4 = 9])$. Calling BDP to solve these bivariate Diophantine equations we obtain the solutions $[\sigma_1, \tau_1] = [54\,{x_1}^2x_2 - 50, 72\,x_1{x_2}^2]$ and $[\sigma_2, \tau_2] = [972\,{x_1}^2x_2 - 482, 1512\,x_1{x_2}^2]$. Hence we have

$$(12\,c_6 + 18\,c_7)\,{x_1}^2x_2 + 9\,c_8 + c_9 = 54\,{x_1}^2x_2 - 50$$
$$(144\,c_6 + 324\,c_7)\,{x_1}^2x_2 + 81\,c_8 + c_9 = 972\,{x_1}^2x_2 - 482$$

Then we solve the Vandermonde linear systems

$$\begin{bmatrix} 12 & 18 \\ 144 & 324 \end{bmatrix} \begin{bmatrix} c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 54 \\ 972 \end{bmatrix} \text{ and } \begin{bmatrix} 9 & 1 \\ 81 & 1 \end{bmatrix} \begin{bmatrix} c_8 \\ c_9 \end{bmatrix} = \begin{bmatrix} -50 \\ -482 \end{bmatrix}$$

to obtain $c_6 = 0, c_7 = 3, c_8 = -6, c_9 = 4$. So $g_1 = 3\,{x_1}^2x_2x_3{x_4}^2 - 6\,{x_4}^2 + 4$. Then by division we get $f_1 = (e_5^{(1)} - f_0g_1)/g_0 = 2\,x_1x_2{x_4}^3 + 8\,x_2{x_3}^4$. Hence

$$f^{(1)} = f_0 + \left(2\,x_1{x_2}^2{x_4}^3 + 6\,x_1{x_2}^2x_4\right)(x_5 - 1)$$
$$g^{(1)} = g_0 + \left(3\,{x_1}^2x_2x_3{x_4}^2 - 6\,{x_4}^2 + 4\right)(x_5 - 1)\,.$$

Note that we use the division step above also as a check for the correctness of the SHL assumption. Since the solution to the MDP is unique, we would have $g_0 \nmid (e_5^{(1)} - f_0g_1)$, if the assumption was wrong.

Now following Lemma 1 by looking at the monomials of f_1 and g_1, we assume that the form of the f_2 and g_2 are

$$f_2 = c_1x_1{x_2}^2{x_4}^3 + c_2x_1{x_2}^2x_4 + c_3$$
$$g_2 = c_4{x_1}^2x_2x_3{x_4}^2 + c_5{x_4}^2 + c_6$$

for some unknowns $C = \{c_1, \ldots, c_6\}$. After computing the next error $a - f^{(1)}g^{(1)}$ we compute $e_5^{(2)}$ and the MDP to be solved is:

$$D := f_0\left(c_4{x_1}^2x_2x_3{x_4}^2 + c_5{x_4}^2 + c_6\right) + g_0\left(c_1x_1{x_2}^2{x_4}^3 + c_2x_1{x_2}^2x_4 + c_3\right) = e_5^{(2)}.$$

We need 2 evaluations again: Choose $[x_3 = 5, x_4 = 6]$ and $[x_3 = 5^2, x_4 = 6^2]$ and compute

$$\begin{aligned} D([x_3 = 5, x_4 = 6]) &:= \left({x_1}^8 + 950\,x_1{x_2}^2 + 30\,{x_2}^2 - 5\right)\left(180\,c_4{x_1}^2x_2 + 36\,c_5 + c_6\right) \\ &\quad + \left({x_1}^8 + 1290\,{x_1}^2x_2 - 104\right)\left(216\,c_1x_1{x_2}^2 + 6\,c_2x_1{x_2}^2 + c_3\right) \\ &= 18\,{x_1}^9{x_2}^2 - 108\,{x_1}^8 + 23220\,{x_1}^3{x_2}^3 - 104472\,x_1{x_2}^2 - 3240\,{x_2}^2 + 540 \end{aligned}$$

and similarly for $D([x_3 = 25, x_4 = 36])$. Calling BDP we obtain the solutions to these bivariate Diophantine equations $[\sigma_1, \tau_1] = [-108, 18\,x_1{x_2}^2]$ and $[\sigma_2, \tau_2] = [-3888, 108\,x_1{x_2}^2]$ respectively. Hence we have $180\,c_4{x_1}^2x_2 + 36\,c_5 + c_6 = -108$ and $32400\,c_4{x_1}^2x_2 + 1296\,c_5 + c_6 = -3888$ respectively. Then we solve the Vandermonde linear systems

$$[180]\,[c_4] = [0] \text{ and } \begin{bmatrix} 36 & 1 \\ 1296 & 1 \end{bmatrix} \begin{bmatrix} c_5 \\ c_6 \end{bmatrix} = \begin{bmatrix} -108 \\ -3888 \end{bmatrix}$$

to obtain $c_4 = 0, c_5 = -3, c_6 = 0$. So $g_2 = -3\,{x_4}^2$. Then by division we get $f_2 = (e_5^{(2)} - f_0 g_2)/g_0 = 3\,x_1{x_2}^2x_4\,(x_5-1)^2$. Hence

$$\begin{aligned} f^{(2)} &= f^{(1)} + 3\,x_1{x_2}^2x_4\,(x_5-1)^2 \\ &= {x_1}^8 + 2\,x_1{x_2}^2{x_4}^3x_5 + 4\,x_1{x_2}^2{x_3}^3 + 3\,x_1{x_2}^2x_4{x_5}^2 + {x_2}^2x_3x_4 - 5 \\ g^{(2)} &= g^{(1)} + \left(-3\,{x_4}^2\right)(x_5-1)^2 \\ &= {x_1}^8 + 3\,{x_1}^2x_2x_3x_4{}^2x_5 + 5\,{x_1}^2x_2{x_3}^2x_4 - 3\,{x_4}^2{x_5}^2 + 4\,x_5. \end{aligned}$$

The next error $e_5^{(3)} = a - f^{(2)}g^{(2)} = 0$ and we have the factors!

We used four evaluations and solved three (2×2) linear systems. For the same problem KHL would need to find 9 linearly independent homogeneous linear equations out of 28 equations first. A natural question is, is it possible for KHL to focus on to some subset of the variables first? The answer is no. The systems of equations constructed by KHL are coupled.

References

1. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties and Algorithms, 3rd ed. Springer, Heidleberg (2007)
2. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer (1992)
3. Kaltofen, E.: Sparse Hensel lifting. Proceedings of EUROCAL '85, LNCS, vol 204, pp. 4–17. Springer (1985)
4. Lee, M.M.: Factorization of multivariate polynomials. Ph.D. Thesis. (2013)
5. Miola, A., Yun, D.Y.Y.: Computational Aspects of Hensel-type Univariate Polynomial Greatest Common Divisor Algorithms. Proceedings of EUROSAM '74, pp. 46–54. ACM Press (1974)
6. Monagan, M.B., Tuncer, B.: Some results on counting roots of polynomials and the Sylvester resultant. To appear in Proceedings of FPSAC 2016. DMTCS (2016)
7. Monagan, M.B., Pearce, R.: POLY: A New Polynomial Data Structure for Maple 17. Computer Mathematics, pp. 325–348. Springer (2014)
8. Schwartz, J.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM, **27**:701–717. ACM Press (1980)
9. Wang, P.S.: An improved Multivariate Polynomial Factoring Algorithm. Mathematics of Computation, **32**:1215–1231. AMS (1978)
10. Wang, P.S., Rothschild, L.P.: Factoring multivariate polynomials over the integers. Mathematics of Computation, **29**(131):935–950. AMS (1975)
11. Yun, D.Y.Y.: The Hensel Lemma in algebraic manipulation. Ph.D. Thesis. (1974)
12. Zippel, R.: Interpolating polynomials from their values. J. Symbolic Comput., **9**:375–403. Academic Press (1990)
13. Zippel, R.E.: Probabilistic algorithms for sparse polynomials. Proceedings of EUROSAM '79, LNCS vol. 72, pp. 216–226. Springer (1979)
14. Zippel, R.E.: Newton's iteration and the sparse Hensel algorithm. Proceedings of SYMSAC '81, pp. 68–72. ACM Press (1981)