Factoring Multivariate Polynomials with Many Factors and Huge Coefficients

Michael Monagan(✉) and Baris Tuncer

Department of Mathematics, Simon Fraser University,
Burnaby, BC V5A 1S6, Canada
{mmonagan,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 recover the multivariate factors from their images using a process known as multivariate Hensel lifting. For the case when the factors are expected to be sparse, at CASC 2016, we introduced a new approach which uses sparse polynomial interpolation to solve the multivariate polynomial diophantine equations that arise inside Hensel lifting.

In this work we extend our previous work to the case when the number of factors to be computed is more than 2. Secondly, for the case where the integer coefficients of the factors are large we develop an efficient p-adic method. We will argue that the probabilistic sparse interpolation method introduced by us provides new options to speed up the factorization for these two cases. Finally we present some experimental data comparing our new methods with previous methods.

Keywords: Polynomial factorization
Sparse polynomial interpolation · Multivariate Hensel lifting
Polynomial diophantine equations

1 Introduction

Suppose we seek to factor a multivariate polynomial $a \in R = \mathbb{Z}[x_1, \ldots, x_n]$. Today many modern computer algebra systems, such as Maple, Magma and Singular, use Wang's incremental design of multivariate Hensel lifting (MHL) to factor multivariate polynomials over integers. MHL was developed by Yun [15] and improved by Wang [13,14].

To factor $a(x_1, \ldots, x_n)$ the first step is to choose a main variable, say x_1, then compute the content of a in x_1 and remove it from a. If $a = \sum_{i=0}^{d} a_i(x_2, \ldots, x_n)x_1^i$, the content of a is $\gcd(a_0, a_1, \ldots, a_d)$, a polynomial in one fewer variables which is factored recursively. Let us assume this has been done.

The second step identifies any repeated factors in a by doing a *square-free factorization*. See Chap. 8 of [2]. In this step one obtains the factorization $a = b_1 b_2^2 b_3^3 \cdots b_k^k$ such that each factor b_i has no repeated factors and $\gcd(b_i, b_j) = 1$.

V. P. Gerdt et al. (Eds.): CASC 2018, LNCS 11077, pp. 319–334, 2018.
https://doi.org/10.1007/978-3-319-99639-4_22

Let us assume this has also been done. So let $a = f_1 f_2 \ldots f_r$ be the irreducible factorization of a over $\mathbb{Z}$. Also, let $\#f$ denote the number of terms of a polynomial f and $\text{Supp}(f)$ denote the support f, i.e., the set of monomials in f.

MHL chooses an evaluation point $\alpha = (\alpha_2, \alpha_3, \ldots, \alpha_n) \in \mathbb{Z}^{n-1}$ where the α_i's are preferably small and contain many zeros. Then $a(x_1, \alpha)$ is factored over $\mathbb{Z}$. The evaluation point α must satisfy

(i) $L(\alpha) \neq 0$ where L is the leading coefficient of a in x_1,
(ii) $a(x_1, \alpha)$ must have no repeated factors in x_1 and
(iii) $f_i(x_1, \alpha)$ must be irreducible over $\mathbb{Q}$.

If any condition is not satisfied the algorithm must restart with a new evaluation point. Conditions (i) and (ii) may be imposed in advance of the next step. One way to ensure that condition (iii) is true with high probability is to pick a second evaluation point $\beta = (\beta_2, \ldots, \beta_n) \in \mathbb{Z}^{n-1}$, factor $a(x_1, \beta)$ over $\mathbb{Z}$ and check that the two factorizations have the same degree pattern before proceeding.

For simplicity let us assume a is monic and suppose we have obtained the monic factors $f_i(x_1, \alpha)$ in $\mathbb{Z}[x_1]$. Next the algorithm picks a prime p which is big enough to cover the coefficients of a and the factors f_i of a.

The input to MHL is $a, \alpha, f_i(x_1, \alpha)$ and p such that $a(x_1, \alpha) = \prod_{i=1}^{r} f_i(x_1, \alpha)$ where $\gcd(f_i(x_1, \alpha), f_j(x_1, \alpha)) = 1$ in $\mathbb{Z}_p[x_1]$ for $i \neq j$. If the gcd condition is not satisfied, the algorithm chooses a new prime p until it is.

There are two main subroutines in the design of MHL. For details see Chap. 6 of [2]. The first one is the leading coefficient correction algorithm (LCC). The most well-known is the Wang's heuristic LCC [14] which works well in practice and is the one Maple currently uses. There are other approaches by Kaltofen [6] and most recently by Lee [9]. In our implementation we use Wang's LCC.

In a typical application of Wang's LCC, one first factors the leading coefficient of a, a polynomial in $\mathbb{Z}[x_2, \ldots, x_n]$, by a recursive call and then one applies LCC before the j^{th} step of MHL. Then the total cost of the factorization is given by the cost of LCC + the cost of factoring $a(x_1, \alpha)$ over $\mathbb{Z}$ + the cost of MHL. One can easily construct examples where LCC or factoring $a(x_1, \alpha)$ dominates the cost. However this is not typical. Usually MHL dominates the cost.

The second main subroutine solves a multivariate polynomial diophantine problem (MDP). In MHL, for each j with $2 \leq j \leq n$, Wang's design of MHL must solve many instances of the MDP in $\mathbb{Z}_p[x_1, \ldots, x_{j-1}]$. Wang's method for solving an MDP (see Algorithm 2) is recursive. Although Wang's method performs significantly better than the previous algorithm that he developed with Rothschild in [14], it does not explicitly take sparsity into account. During computation, the ideal-adic representation of factors is dense when the evaluation points $\alpha_2, \ldots, \alpha_n$ are non-zero. In practice, conditions (i) and (iii) of LCC may force many non-zero α_j's. This makes Wang's approach exponential in n.

Zippel's sparse interpolation [18] was the first probabilistic method aimed at taking sparsity into account. Based on sparse interpolation and multivariate Newton's iteration, Zippel then introduced a sparse Hensel lifting (ZSHL) algorithm in [17,19], which uses a MHL organization different from Wang's.

Another approach for sparse Hensel lifting for the sparse case was proposed by Kaltofen (KSHL) in [6]. Kaltofen's method is also based on Wang's incremental design of MHL but it uses a LCC different from Wang's LCC and offers a distinct solution to the multivariate diophantine problem (MDP) that appears in Wang's design of MHL.

At CASC 2016 the authors proposed a new practical sparse Hensel Lifting algorithm (MTSHL) [11]. It is also based on Wang's incremental design of MHL and LCC but offers a solution to the MDP different from those of Zippel and Kaltofen. To solve the MDP problem appearing in MHL, MTSHL exploits the fact that at each step of MHL, the solutions to MDP's, which are just Taylor polynomial coefficients, are structurally related. At the jth step of MHL we are recovering x_j in the factors. Let f be one such factor in $\mathbb{Z}_p[x_1, x_2, \ldots, x_j]$ and let $f = \sum_{k=0}^{l} f_k(x_j - \alpha_j)^k$ be its Taylor representation. At this point we know only f_0. But $\text{Supp}(f_k) \subseteq \text{Supp}(f_{k-1})$ with high probability if α_j is chosen randomly from $[0, p-1]$ and p is sufficiently large. MTSHL is built on this key observation.

In this paper we consider the case where a has $r > 2$ factors and secondly the case where the factors have large integer coefficients. When $r > 2$, the MDP problem is called a multiterm MDP problem and an approach to the solution to this problem is described in [2]. It reduces the multiterm MDP problem to $r-1$ two term MDP problems. Our previous implementation of MTSHL described in [11] also used this approach.

In Sect. 2 we define the MDP problem in the context of MHL. See Algorithms 1 and 2. In Sect. 3 we discuss main ideas for the solution to the MDP used by MTSHL and present it as Algorithm 3 to make our explanation precise. We call Algorithm 3 MTSHL-d (d stands for direct), since it differs from our previous version of MTSHL (Algorithm 4 in [11]) in how it solves MDP problems when $r > 2$. For $r = 2$ it is the same as Algorithm 4 in [11].

In Sect. 4 we discuss the case $r > 2$. We argue that the probabilistic sparse interpolation method used in the design of MTSHL allows us to reduce the time spent solving multiterm MDP's by up to a factor of $r - 1$. Because our proposal also reduces the multiplication cost in the previous approach described in [2], the observed speedup is sometimes greater than $r - 1$.

In Sect. 5, we study the case where the integer coefficients of the factors are large. The current approach (see [2]) chooses a prime p and $l > 0$ such that p^l bounds any coefficients in the factors f_i of a. We show that the sparse MDP solver developed in [11] renders an improved option. Suppose one factor $f \in \mathbb{Z}[x_1, \ldots, x_n]$ has a p-adic representation $f = \sum_{k=0}^{l} f_k p^k$. We show that in this case also $\text{Supp}(f_k) \subseteq \text{Supp}(f_{k-1})$ with high probability if p is chosen randomly. Therefore we propose first to factor a in $\mathbb{Z}_p[x_1, \ldots, x_n]$ by doing all arithmetic mod p where p is a machine prime (e.g. 63 bits on a 64 bit computer), i.e. run the entire Hensel lifting modulo a machine prime. Then lift the solution to $\mathbb{Z}_{p^l}[x_1, \ldots, x_n]$ by computing f_k, again by solving each MDP appearing in the lifting process using the sparse interpolation developed in the design of MTSHL. Using this approach most of the computation is modulo p a machine prime.

In Sect. 6 we present some timing data to compare our new approaches with previous approaches and end with some concluding remarks.

In the paper we assume the input polynomial a is monic in x_1 so as not to complicate the presentation with LCC. We note that what we explain remains true for the non-monic case with slight modifications. Our implementation uses Wang's LCC for the non-monic case.

2 The Multivariate Diophantine Problem (MDP)

The Multivariate Diophantine Problem (MDP) arises naturally as a subproblem of the incremental design of MHL developed by Wang. For completeness we provide the j^{th} step of MHL as Algorithm 1 for the monic case and Wang's solution to the MDP as Algorithm 2.

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

Input : $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$, $f_{j-1,1}, \ldots, f_{j-1,r} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ where $a_j, f_{j-1,i}$ are monic in x_1 and $a_j(x_j = \alpha_j) = \prod_{i=1}^{r} f_{j-1,i}$.
Output : $f_{j,1}, \ldots, f_{j,r} \in \mathbb{Z}_p[x_1, \ldots, x_j]$ such that $f_{j,i}(x_j = \alpha_j) = f_{j-1,i}$ and $a_j = \prod_{i=1}^{r} f_{j,i}$ or FAIL.

1: **for** i from 1 to r **do**
2: $\quad \sigma_{0,i} \leftarrow f_{j-1,i}$; $f_{j,i} \leftarrow \sigma_{0,i}$; $b_{j,i} \leftarrow \prod_{k=1,k\neq i}^{r} f_{j-1,k}$.
3: **end for**
4: $error \leftarrow a_j - \prod_{i=1}^{r} f_{j,i}$.
5: **for** k from 1 while $error \neq 0$ and $\sum_{i=1}^{r} \deg_{x_j} f_{j,i} < \deg_{x_j} a_j$ **do**
6: $\quad c_k \leftarrow$ Taylor coefficient of $(x_j - \alpha_j)^k$ of $error$ at $x_j = \alpha_j$
7: $\quad$ **if** $c_k \neq 0$ **then**
8: $\quad\quad$ Solve $\text{MDP}_{j,k}$: $\sigma_{k,1} b_{j,1} + \cdots + \sigma_{k,r} b_{j,r} = c_k$ for $\sigma_{k,i} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$.
9: $\quad\quad$ **for** i from 1 to r **do**
10: $\quad\quad\quad f_{j,i} \leftarrow f_{j,i} + \sigma_{k,i} \times (x_j - \alpha_j)^k$
11: $\quad\quad$ **end for**
12: $\quad\quad error \leftarrow a_j - \prod_{i=1}^{r} f_{j,i}$.
13: $\quad$ **end if**
14: **end for**
15: **if** $error = 0$ **then** return $f_{j,1}, \ldots, f_{j,r}$ **else** return FAIL **end if**

The MDP appears at line 8 of Algorithm 1. Consider the case where the number of factors r to be computed is 2, i.e., $r = 2$. We discuss the case $r > 2$ in Sect. 4.

Let $u, w, c \in \mathbb{Z}_p[x_1, \ldots, x_j]$ with u and w monic with respect to the variable x_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 $\alpha_i \in \mathbb{Z}$. The MDP is to find multivariate polynomials $\sigma, \tau \in \mathbb{Z}_p[x_1, \ldots, x_j]$ that satisfy

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

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

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

It can be shown that the solution (σ, τ) exists and is unique and 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 Chap. 2 of [2]).

To solve the MDP for $j > 1$, Wang uses the same approach as for Hensel Lifting, that is, an ideal-adic lifting approach. See Algorithm 2.

Algorithm 2. WMDS (Wang's multivariate diophantine solver)

Input A point $\alpha_j \in \mathbb{Z}_p$, polynomials $c,\ f_{j,k} \in \mathbb{Z}_p[x_1, \ldots, x_j]$ for $k = 1, \ldots, r$ and an ideal $I = \langle x_2 - \alpha_2, \ldots, x_n - \alpha_n \rangle$ with $n \geq j$ where $\gcd(f_{j,k} \mod I, f_{j,l} \mod I) = 1$ in $\mathbb{Z}_p[x_1]$ for $k \neq l$ and a degree bound d_j satisfying $d_j \geq \max(\deg_{x_j} \sigma_k)$ for $2 \leq i \leq n$. (One may use $d_j = \deg_{x_j} a$)

Output $(\sigma_1, \ldots, \sigma_r) \in \mathbb{Z}_p[x_1, \ldots, x_j]$ satisfying $\sum_{k=1}^{r} \sigma_k b_k = c \in \mathbb{Z}_p[x_1, \ldots, x_j]$ where $b_k = \prod_{i \neq k}^{r} f_{j,i}$ and $\deg_{x_1} \sigma_k < \deg_{x_1} f_{j,k}$ or FAIL if no such solution exists.

1: $b_k \leftarrow \prod_{i \neq k}^{r} f_{j,i}$ for $k = 1, \ldots, r$
2: **if** $j = 1$ **then** use extended Euclidean algorithm (see Ch 2 of [2] for $r = 2$ and Section 4 for $r > 2$) **end if**
3: $(\sigma_{1,0}, \ldots, \sigma_{r,0}) \leftarrow \mathrm{WMDS}(f_{j,k}(x_j = \alpha_j),\ c(x_j = \alpha_j),\ I)$
4: **if** WMDS output FAIL **then** return FAIL **end if**
5: $\sigma_k \leftarrow \sigma_{k,0}$ for $k = 1, \ldots, r$; $error \leftarrow c - \sum_{k=1}^{r} \sigma_k b_k$
6: **for** $i = 1, 2, \ldots, d_j$ **while** $error \neq 0$ **do**
7: $c_i \leftarrow$ Taylor coeff$(error,\ (x_j - \alpha_j)^i)$
8: **if** $c_i \neq 0$ **then**
9: $(s_1, \ldots, s_r) \leftarrow \mathrm{WMDS}(\sigma_k,\ c_i,\ I)$
10: **if** WMDS output FAIL **then** return FAIL **end if**
11: $\sigma_k \leftarrow \sigma_k + s_k \times (x_j - \alpha_j)^i$ for $k = 1, \ldots, r$.
12: $error \leftarrow error - \sum_{k=1}^{r} \sigma_k b_k$
13: **end if**
14: **end for**
15: **if** $error = 0$ **then** return $(\sigma_1, \ldots, \sigma_r)$ **else** return FAIL **end if**

In general, if $\alpha_j \neq 0$ the Taylor series expansion of σ and τ about $x_j = \alpha_j$ is dense in x_j so the $c_i \neq 0$. Then the number of calls to the Euclidean algorithm of Wang's solution to MDP is exponential in n. It is this exponential behaviour that the design of MTSHL eliminates. On the other hand, if MHL can choose some α_j to be 0, for example, if the input polynomial $a(x_1, \ldots, x_n)$ is monic in x_1 then this exponential behaviour may not occur for sparse f and g.

3 MTSHL's Solution to the MDP via Sparse Interpolation

We consider whether we can interpolate $x_2, \ldots, x_j$ in σ and τ in (1) using sparse interpolation methods. If $\beta \in \mathbb{Z}_p$ with $\beta \neq \alpha_j$, then

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

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 $\sigma(x_j = \beta)$ iff $G_j = 1$. However $G_j \neq 1$ is possible. Let $R = \mathrm{res}_{x_1}(u, w)$ be the Sylvester resultant of u and w taken in x_1. Since u, w are monic in x_1 one has[1]

$$G_j \neq 1 \Longleftrightarrow \mathrm{res}_{x_1}(u \bmod K_j,\ w \bmod K_j) = 0 \Longleftrightarrow R(\alpha_2, \ldots, \alpha_{j-1}, \beta) = 0.$$

Let $r = R(\alpha_2, \ldots, \alpha_{j-1}, x_j) \in \mathbb{Z}_p[x_j]$ so that $R(\alpha_2, \ldots, \alpha_{j-1}, \beta) = r(\beta)$. Also $\deg(R) \leq \deg(u)\deg(w)$ [1]. Now if β is chosen at random from $\mathbb{Z}_p$ and $\beta \neq \alpha_j$ then

$$\Pr[G_j \neq 1] = \Pr[r(\beta) = 0] \leq \frac{\deg(r, x_j)}{p-1} \leq \frac{\deg(u)\deg(w)}{p-1}.$$

This bound for $\Pr[G_j \neq 1]$ is a worst case bound. In [10] we show that the average probability for $\Pr[G_j \neq 1] = 1/(p-1)$. Thus if p is large, the probability that $G_j = 1$ is high. Interpolation is thus an option to solve the MDP.

As can be seen from line 10 of Algorithm 1, the solutions to the MDP are the Taylor coefficients of the factors to be computed at the j^{th} step. As such, if $\sigma_{0,i}$ is sparse then the $\sigma_{k,i}$ are also sparse. In line 5 of Algorithm 1, as k increases, on average, the number of terms of the $\sigma_{k,i}$ decrease even for dense cases. That is, on average $\#\sigma_{k,i} < \#\sigma_{k-1,i}$. A natural idea then is to use sparse interpolation techniques to solve the MDP. However, the sparse technique proposed by Zippel [16] is also iterative; it recovers x_2 then x_3 etc. To make one more step in this direction consider the following Lemma whose proof can be found in [11].

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

$$\Pr[\mathrm{Supp}(b_{j+1}) \not\subseteq \mathrm{Supp}(b_j)] \;\leq\; |\mathrm{Supp}(b_{j+1})| \, \frac{d_n - j}{p - d_n + j + 1} \text{ for } 0 \leq j < d_n.$$

Lemma 1 says that for the sparse case, if p is big enough then the probability of $\mathrm{Supp}(b_{j+1}) \subseteq \mathrm{Supp}(b_j)$ is high. This observation suggests, during MHL we use $\sigma_{k-1,i}$ as a form of the solution of $\sigma_{k,i}$. That is, the solutions to the MDP's are related. During MHL, these problems shouldn't be treated independently as previous approaches do. In light of the key role this assumption plays at

[1] This argument also works for the non-monic case if the leading coefficients of u and w w.r.t. x_1 do not vanish at $(\alpha_2, \ldots, \alpha_n)$ modulo p, conditions which we note are imposed by Wang's LCC.

each MHL step $j > 1$, for each factor f_i, we call this assumption $\text{Supp}(\sigma_{k,i}) \subseteq \text{Supp}(\sigma_{k-1,i})$ for all $k > 0$ the **strong SHL assumption**.

Algorithms 3 and 4 below show how this assumption can be combined with the sparse interpolation idea of Zippel [16] to reduce the solution to the MDP problem to solving linear systems over $\mathbb{Z}_p$. To see how MTSHL works on a concrete example for $r = 2$ and how MTSHL decreases the evaluation cost that sparse interpolation brings see [11].

We present the j^{th} step of the new version of MTSHL in Algorithm 4 below and call it as MTSHL-d, as a shortcut for MTSHL direct. For $r = 2$ MTSHL-d is equivalent to MTSHL described in [11]. In the following section we discuss the case $r > 2$ and make it clear why we call it MTSHL direct.

4 The Multiterm Diophantine Problem

Let the input polynomial $a(x_1, \ldots, x_n)$ be square-free with total degree d and irreducible factorization of a be

$$a = f_1 \cdots f_r \in \mathbb{Z}[x_1, \ldots, x_n].$$

We consider the case $r > 2$. We start with the unique factorization of $a_1(x_1) = a(x_1, \alpha) = u_1(x_1) \cdots u_r(x_1) \in \mathbb{Z}[x_1]$. By Hilbert's irreducibility theorem [7] most probably $u_i(x_1) = f_i(x_1, \alpha)$. Next we choose a prime p which is big enough to cover the coefficients occurring in each f_i and then pass to mod p

$$a(x_1, \alpha) = u_1(x_1) \cdots u_r(x_1) \in \mathbb{Z}_p[x_1].$$

We need $\gcd(u_i, u_j) = 1 \in \mathbb{Z}_p[x_1]$ for all $1 \leq i < j \leq r$. Otherwise we choose a different prime and repeat the process.

Suppose $f_i = \sum_{k=0} \sigma_{i,k}(x_j - \alpha_j)^k$. So $\sigma_{i,k}$ is the k^{th} Taylor coefficient of the i^{th} factor to be computed in the j^{th} step of MHL. (See line 10 of Algorithm 1.) During the j^{th} step of MHL, for each iteration $k > 0$, the algorithm computes $\sigma_{k,i}$, by solving the multiterm Diophantine problem (multi-MDP), which is a natural generalization of the MDP defined in Sect. 2 and denoted as $\text{MDP}_{j,k}$ in line 8 of Algorithm 1. It has the form

$$\text{MDP}_{j,k}: \ \sigma_{k,1}b_1 + \cdots + \sigma_{k,r}b_r = c_k,$$

where $b_k = \prod_{i=1, i\neq k}^{r} f_{j-1,i}(x_1, \ldots, x_{j-1})$. So, given b_k and c_k in $\mathbb{Z}_p[x_1, \ldots, x_{j-1}]$, the goal is to find $\sigma_{k,i}$ for each i.

The current approach to solve a multiterm MDP is to reduce it into $r - 1$ two term MDP's. We describe the idea with an example. Let $r = 4$ and to save some space let $u_i = f_{j-1,i}$. Then

$$\begin{aligned} c_k &= \sigma_{k,1}b_1 + \sigma_{k,2}b_2 + \sigma_{k,3}b_3 + \sigma_{k,4}b_4 \\ &= \sigma_{k,1}u_2u_3u_4 + \sigma_{k,2}u_1u_3u_4 + \sigma_{k,3}u_1u_2u_4 + \sigma_{k,4}u_1u_2u_3 \\ &= \sigma_{k,1}u_2u_3u_4 + u_1(\sigma_{k,2}u_3u_4 + u_2(\sigma_{k,3}u_4 + \sigma_{k,4}u_3)). \end{aligned}$$

Algorithm 3. SparseInt: solve an MDP using a sparse interpolation

Input: Polynomials $f_i, \sigma_i, c \in \mathbb{Z}_p[x_1, x_2, \ldots, x_{j-1}]$ for $i = 1, \ldots, r$. f_i are monic in x_1 and p a prime.

Output: Update σ_i so that they form a solution to the multi-MDP $\sigma_1 b_1 + \cdots + \sigma_r b_r = c$ in $\mathbb{Z}_p[x_1, x_2, \ldots, x_{j-1}]$ where $b_i = \prod_{k=1, k\neq i}^{r} f_i$ or FAIL.

1: **for** i from 1 to r **do**
2: $\sigma_i \leftarrow \sum_{l,k} c_{ilk}(x_3, ..., x_{j-1}) x_1^l x_2^k$ where $c_{ilk} = \sum_{w=1}^{s_{ilk}} c_{ilkw} M_{ilkw}$ with c_{ilkw} are unknown coefficients to be solved for and $x_1^l x_2^k M_{ilkw}$ are the monomials in $\mathrm{Supp}(\sigma_i)$.
3: **end for**
4: Let $t = \max_{i=1}^{r}\{\max s_{ilk} = \max \#c_{ilkw}\}$
5: Pick $(\beta_3, \ldots \beta_{j-1}) \in (\mathbb{Z}_p \backslash \{0\})^{j-3}$ at random.
6: **for** s from 1 to t **do** (*Precomputation.(see [11])*)
7: Let $Y_s = (x_3 = \beta_3^s, \ldots, x_{j-1} = \beta_{j-1}^s)$.
8: Evaluate $c(x_1, x_2, Y_s)$ and $f_i(x_1, x_2, Y_s)$ for $1 \leq i \leq r$.
9: **end for**
10: **for** i from 1 to r **do**
11: Compute $b_i(x_1, x_2, Y_i) = \prod_{k=1, k\neq i}^{r} f_i(x_1, x_2, Y_i)$ in $\mathbb{Z}_p[x_1, x_2]$.b
12: **end for**
13: **for** i from 1 to r **do**
14: Compute monomial evaluation sets for σ_i

$$\{S_{ilk} = \{m_{ilkw} = M_{ilkw}(\beta_3, \ldots, \beta_{j-1}) : 1 \leq w \leq s_{ilk}\} \text{ for each } l, k\}.$$

15: If $|S_{ikl}| \neq s_{ikl}$ for some ikl try a different choice for $(\beta_3, \ldots, \beta_{j-1})$.
16: If this fails, **return** FAIL. (p *is not big enough*)
17: Let $t_i = \max_{l,k} s_{ilk}$
18: **for** s from 1 to t_i **do** (*Compute the bivariate images of* σ_i)
19: Solve $\tilde{\sigma}_1(x_1, x_2) f_1(x_1, x_2, Y_i) + \cdots + \tilde{\sigma}_r(x_1, x_2) f_r(x_1, x_2, Y_i) = c(x_1, x_2, Y_i)$ in $\mathbb{Z}_p[x_1, x_2]$ for $\tilde{\sigma}_i(x_1, x_2)$ using multi-BDP (see section 4).
20: **if** multi-BDP returns FAIL **then return** FAIL **end if**
(multi-BDP fails if it choses $\gamma \in \mathbb{Z}_p$ with $\gcd(f_i(x_1, \gamma, Y_i), f_j(x_1, \gamma, Y_i)) \neq 1$ for some $i \neq j$).
21: **end for**
22: **for** each l, k **do**
23: Construct and solve the $s_{ilk} \times s_{ilk}$ linear system

$$\left\{ \sum_{w=1}^{s_{ilk}} c_{ilkw}\, m_{ilkw}^n \;=\; \text{coefficient of } x_1^l x_2^k \text{ in } \tilde{\sigma}_i(x_1, x_2) \;\text{ for } 1 \leq n \leq s_{ilk} \right\}$$

for the coefficients c_{ilkw} of $c_{ilk}(x_3, \ldots, x_{j-1})$. Because it is a Vandermonde system in m_{iklw} which are distinct by Step 15 it has a unique solution.
24: **end for**
25: Substitute the solutions for c_{ilkw} into σ_i
26: **end for**
27: Verify probabilistically whether $\sum_{i=1}^{r} \sigma_i b_i = c$:
Pick $\beta = (\beta_1, \ldots \beta_{j-1}) \in \mathbb{Z}_p^{j-1}$ at random.
if $\sum_{i=1}^{r} \sigma_i(\beta) b_i(\beta) \neq c(\beta)$ **then return** FAIL **end if**
28: **return** $\sigma_1, \ldots, \sigma_r$

Algorithm 4. j^{th} step of MTSHL-d for $j > 1$.

Input : $\alpha_j \in \mathbb{Z}_p$, $a_j \in \mathbb{Z}_p[x_1, \ldots, x_j]$, $f_{j-1,1}, \ldots, f_{j-1,r} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$ where $a_j, f_{j-1,i}$ are monic in x_1 and $a_j(x_j = \alpha_j) = \prod_{i=1}^{r} f_{j-1,i}$.
Output : $f_{j,1}, \ldots, f_{j,r} \in \mathbb{Z}_p[x_1, \ldots, x_j]$ such that $f_{j,i}(x_j = \alpha_j) = f_{j-1,i}$ and $a_j = \prod_{i=1}^{r} f_{j,i}$ or FAIL.

1: **for** i from 1 to r **do** $f_{j,i} \leftarrow f_{j-1,i}$, $\sigma_{0,i} \leftarrow f_{j-1,i}$ **end do**
2: $error \leftarrow a_j - \prod_{i=1}^{r} f_{j,i}$
3: **for** $k = 1, 2, 3, \ldots$ **while** $error \neq 0$ and $\sum_{i=1}^{r} \deg(f_{j,i}, x_j) < \deg(a_j, x_j)$ **do**
4: $c_k \leftarrow$ Taylor coefficient of $(x_j - \alpha_j)^k$ of $error$ at $x_j = \alpha_j$
5: **if** $c_k \neq 0$ **then**
6: Solve the $\text{MDP}_{\text{j,k}}$ (see line 8 of Alg. 1) without computing $b_{j,i}$ as follows:
7: **for** i from 1 to r **do** $\sigma_{k,i} \leftarrow \sigma_{k-1,i}$ **end do** (Strong SHL assumption.)
8: $(\sigma_{k,1}, \ldots, \sigma_{k,r}) \leftarrow$ **SparseInt**($f_{j-1,i}$, c_k, $\sigma_{k,i}$, $i = 1, \ldots, r$) (see Alg. 3)
9: **if** $(\sigma_{k,1}, \ldots, \sigma_{k,r})$=FAIL **then** restart MTSHL-d with a new α **end if**
10: **for** i from 1 to r **do** $f_{j,i} \leftarrow f_{j,i} + \sigma_{k,i} \times (x_j - \alpha_j)^k$ **end do**
11: $error \leftarrow a_j - \prod_{i=1}^{r} f_{j,i}$
12: **end if**
13: **end for**
14: **if** $error = 0$ **then** return $f_{j,1}, \ldots, f_{j,r}$ **else** return FAIL **end if**

We first solve the MDP $\sigma_{k,1}u_2u_3u_4 + u_1w_1 = c_k$ for $\sigma_{k,1}$ and w_1. Then we solve $\sigma_{k,2}u_3u_4 + u_2w_2 = w_1$ for $\sigma_{k,2}$ and w_2. Finally we solve $\sigma_{k,3}u_4 + \sigma_{k,4}u_3 = w_2$ to compute $\sigma_{k,3}$ and $\sigma_{k,4}$. Let us call this approach as the iterative approach to solve the multiterm MDP.

Note that Wang's approach to solve the MDP is recursive. So when $r > 2$, the iterative approach to solve multiterm MDP makes Wang's design highly recursive. Also, if the polynomials u_i have many terms then the b_i's will be large and expensive to compute. If we use the probabilistic sparse MDP solver of MTSHL as described in [11] for each of these MDP's, then we will first compute the b_i's and then evaluate b_i's at random points. But evaluation is one of the most costly operations in sparse interpolation and this cost increases as the size of the polynomial to be evaluated increases.

However, the probabilistic non-recursive sparse interpolation idea used to solve the MDP's in MHL renders another simple and efficient option. One can invoke the sparse MDP solver to compute the $\sigma_{k,i}$'s simultaneously without reducing $\text{MDP}_{j,k}$ to $r-1$ two term MDP's in the following way.

According to Lemma 1, if α_j is random and p is big, then for each factor $f_{j,i}$, with probability $\geq 1 - |\text{Supp}(\sigma_{k,i})| \frac{d_i - i}{p - d_i + j + 1}$ one has $\text{Supp}(\sigma_{k,i}) \subseteq \text{Supp}(\sigma_{k-1,i})$ for $k = 1, .., d_i$ where $\sigma_{0,i}$ is defined as $\sigma_{0,i} := f_{j-1,i}$ and $d_i = \deg_{x_j}(f_{j,i})$. Therefore to solve $\text{MDP}_{j,k}$ we use $\text{Supp}(\sigma_{k-1,i})$ as a skeleton of the solution of $\sigma_{k,i}$. That is, if $\sigma_{k-1,i} = \sum_{l,k} m_{ilk} M_{ilk}$ for $m_{ilk} \in \mathbb{Z}_p - \{0\}$ with distinct monomials in $M_{ilk} \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$, then we construct $\bar{\sigma}_{k,i} = \sum_{l,k} c_{ilk} M_{ilk}$ as a solution form (skeleton) of $\sigma_{k,i}$, where c_{ilk} are to be computed.

At the k^{th} iteration suppose that we need t_i evaluations to recover the coefficients c_{ilk} (see line 17 of Algorithm 3). Let $\beta = (\beta_2, \ldots \beta_{j-1})$ where $\beta_i \in \mathbb{Z}_p - \{0\}$ be a random evaluation point. Consider the t_i consecutive univariate multiterm MDP's

$$\tilde{\sigma}_{k,1} b_1(x_1, \beta^s) + \cdots + \tilde{\sigma}_{k,r} b_r(x_1, \beta^s) = c_i(x_1, \beta^s) \text{ for } 1 \le s \le t_i, \tag{2}$$

where the $\tilde{\sigma}_{k,i}$ are to be computed. By uniqueness of the solutions to the multiterm MDP, with average probability $\binom{r}{2}\frac{1}{p}$ one has $\tilde{\sigma}_{k,i} = \sigma_{k,i}(x_1, \beta^s)$.

Equation 2 can be solved efficiently for $\tilde{\sigma}_{k,i}$ using the iterative approach in the univariate domain $\mathbb{Z}_p[x_1]$. Next the univariate images $\bar{\sigma}_{k,i}(x_1, \beta^s)$ of $\bar{\sigma}_{k,i}$ are used to compute the coefficient c_{ilk} of $\bar{\sigma}_{k,i}$ by solving Vandermonde systems which are constructed by equating the coefficients of $\sigma_{k,i}(x_1, \beta^j)$ and $\tilde{\sigma}_{k,i}$ (see line 23 of Algorithm 3). Again, if the strong SHL assumption is true, then by following Zippel's analysis in [16], one can show that with probability $\geq 1 - \frac{(\#f_i)^2}{2(p-1)}$, we have a unique solution to Vandermonde systems.

At this stage we have candidate solutions $\bar{\sigma}_{k,i}$ for the actual solutions $\sigma_{k,i}$ of $\text{MDP}_{j,k}$. Because our assumption $\text{Supp}(\sigma_{k,i}) \subseteq \text{Supp}(\sigma_{k-1,i})$ may be false, we need to verify if $\bar{\sigma}_{k,i} = \sigma_{k,i}$. We do this using a random evaluation in line 27 of Algorithm 3.

What does this approach bring us? First, MTSHL-d essentially follows MTSHL but eliminates an iteration at the cost of an increase in the probability of failure. However this probability is negligible if p is big enough. In our implementation we used a 31 bit prime and MTSHL-d never failed. Since it is an iteration on r, we expect MTSHL-d to solve multi-MDP's faster than MTSHL by a factor of $\mathcal{O}(r)$. This is verified by the experimental data in Table 1 of Sect. 6.

Second, $b_k(x_1, \beta^s) = \prod_{i=1, i\neq k}^{r} f_i(x_1, \beta^s)$, so we don't need to compute $b_k \in \mathbb{Z}_p[x_1, \ldots, x_{j-1}]$. All we need to do is to compute and multiply their univariate images $f_i(x_1, \beta^s)$ of f_i to obtain $b_k(x_1, \beta^j)$.

Finally in MTSHL-d, like MTSHL, we may evaluate down to $\mathbb{Z}[x_1, x_2]$ instead of $\mathbb{Z}[x_1]$ to decrease the number of evaluations t_i needed and the size of the Vandermonde systems (Line 17 in Algorithm 3). To do this MTSHL-d uses multi-Bivariate Diophant Solver (multi-BDP). We implemented Multi-BDP in C. It solves the bivariate multi-MDP by the iterative approach and uses evaluation and interpolation on x_2 to reduce to the univariate case.

5 The Case Modulo p^l with $l > 1$

When the integer coefficients of a or the factors of a to be computed are huge the current strategy implemented by most of the computer algebra platforms, including Maple, Singular [9] and Magma [12], is the following. For details see [2]. First we pick a prime p and a natural number $l > 0$ such that the ring $\mathbb{Z}_{p^l}$ can be identified with the ring $\mathbb{Z}$. That is, we find a bound B such that the integer coefficients of the polynomial a to be factored and its irreducible factors are bounded by B. One way to choose such an upper bound B is given by [4]. Then

Algorithm 5. LiftTheFactors for $r = 2$ (optimized)

Input : $a \in \mathbb{Z}[x_1, \ldots, x_n]$, $f_0, g_0 \in \mathbb{Z}_p[x_1, \ldots, x_n]$ where a, f_0, g_0 are monic in x_1 and $a = f_0 g_0$ in $\mathbb{Z}_p[x_1, \ldots, x_n]$. Also an integer bound $l > 0$ (For example, [Lemma 14, [4]]).

Output : $f, g \in \mathbb{Z}[x_1, \ldots, x_j]$ such that $a = fg \in \mathbb{Z}[x_1, \ldots, x_n]$ or FAIL

1: $(f,\ g) \leftarrow (\text{mods}(u_0, p),\ \text{mods}(w_0, p))$. (# *use symmetric range*)
2: *modulus* $\leftarrow 1$.
3: *error* $\leftarrow (a - fg)/p$, $(\sigma_f, \sigma_g) \leftarrow (f, g)$
4: **for** i from 1 to l **while** *error* $\neq 0$ **do**
5: *modulus* $\leftarrow$ *modulus* $\times\ p$, $c \leftarrow$ *error* mod p
6: # *Solve the MDP* $\sigma\, u_0 + \tau\, w_0 = c$ *for* σ *and* τ *in* $\mathbb{Z}_p[x_1, \ldots, x_n]$*:*
7: $(\sigma, \tau) \leftarrow$ **SparseInt**$(f, g, \sigma_f, \sigma_g, c)$ (Algorithm 3)
8: **if SparseInt** output FAIL **then return** FAIL **end if**
9: $(\sigma, \tau) \leftarrow (\text{mods}(\sigma, p),\ \text{mods}(\tau, p))$. (# *use symmetric range*)
10: $(\sigma_f, \sigma_g) \leftarrow (\sigma, \tau)$, *error* $\leftarrow$ $(error - (f\tau + g\sigma) + \sigma\tau \times modulus)/p$
11: $(f, g) \leftarrow (f + \sigma \times modulus,\ g + \tau \times modulus)$.
12: **end for**
13: **if** *error* $\neq 0$ **then return** FAIL **else return** (f, g) **end if**

we choose l such that $p^l > 2B$. Next the MDP solution in $\mathbb{Z}_p[x_1]$ is lifted to the solution in $\mathbb{Z}_{p^l}[x_1]$. The second step is to lift the solution from $\mathbb{Z}_{p^l}[x_1]$ to $\mathbb{Z}_{p^l}[x_1, \ldots, x_n]$. Note that in the second step all arithmetic is in $\mathbb{Z}_{p^l}$ with $p^l > 2B$. In this section we question whether this strategy is the best approach for the case $l > 1$.

Suppose for example that the coefficients of the factors are bounded by p^{10}. Before the factorization we don't have this information. Since most likely the coefficient bound $B > p^{20}$, this means that throughout MHL all integer arithmetic is modulo p^{20} which is expensive.

MTSHL's sparse multivariate diophantine solver allows us to propose an approach that eliminates most of the multi-precision arithmetic and allows us to lift up to the size of the actual coefficients in the factors, thus avoiding B.

- First choose a random $(m+1)$-bit machine prime p, i.e. $p \in [2^m < p < 2^{m+1}]$ and compute the factorization of a by lifting the factorization in $\mathbb{Z}_p[x_1]$ to in $\mathbb{Z}_p[x_1, \ldots, x_n]$ with MTSHL-d. Most of this work is mod p.
- Next compute a lifting bound B. One may use Lemma 14 of [4] for this purpose. Now pick the smallest l such that $p^l > 2B$.
- Then as a second stage do a p-adic lift of the factorization from $\mathbb{Z}_p[x_1, \ldots, x_n]$ stopping when f and g are recovered or we exceed p^l. The p-adic lift is presented as Algorithm 5. It reduces to solving MDPs in $\mathbb{Z}_p[x_1, \ldots, x_n]$.

To make the following explanation easier we assume $r = 2$ and suppose that $a = uw$ where $a, u, w \in \mathbb{Z}[x_1, \ldots, x_n]$ and u, w are unknown to us. As a first step we choose an evaluation ideal $I = \langle x_2 - \alpha_2, \ldots, x_n - \alpha_n \rangle$ with randomly chosen α_i from $[0, p-1]$ such that conditions (i) and (ii) for MHL are satisfied with $l = 1$. Then there is a factorization $a = u^{(n)} w^{(n)} \in \mathbb{Z}_p[x_1, \ldots, x_n]$. This factorization is computed using MTSHL-d.

Now suppose that u (similarly w) has the form

$$u = \sum_{j=1}^{t} c_j M_j(x_1, \ldots, x_n) = \sum_{j=1}^{t} \sum_{i=0}^{l-1} s_{ji} p^i M_j(x_1, \ldots, x_n),$$

where the M_j are distinct monomials and $0 \neq c_j \in \mathbb{Z}$ with $c_j = \sum_{i=0}^{l-1} s_{ji} p^j$ where $-p^l/2 < s_{ji} < p^l/2$. Then we have

$$u = \sum_{i=0}^{l-1} \left(\sum_{j=1}^{t} s_{ji} M_j(x_1, \ldots, x_n) \right) p^i = \sum_{i=0}^{l-1} u_i p^i.$$

It follows that

$$\frac{u - \sum_{i=0}^{k-1} u_i p^i}{p^k} = \sum_{j=1}^{t} \left(\sum_{i=k}^{l-1} s_{ji} p^{i-k} \right) M_j(x_1, \ldots, x_n).$$

Also, we have $u_0 = u \mod p \neq 0$ since in the first stage u is lifted from u_0. Now we make a key observation: If p is chosen at random such that $2^m < p < 2^{m+1}$, the probability that $p \mid c_i$ is $\Pr[p \mid c_i] = \frac{\#\text{distinct (m+1)bit prime divisors of } c_i}{\#\text{m bit primes}}$. Let $\pi(s)$ be the number of primes $\leq s$. Since there are at most $\lfloor \log_{2^m}(c_i) \rfloor$ many $(m+1)$-bit primes dividing c_i we have

$$\Pr[p \mid c_i] \leq \frac{\lfloor \log_{2^m}(c_i) \rfloor}{\pi(2^{m+1}) - \pi(2^m)} \leq \frac{l}{\pi(2^{m+1}) - \pi(2^m)}$$

This probability is very small because according to the prime number theorem $\pi(s) \sim s/\log(s)$ and hence $\pi(2^{m+1}) - \pi(2^m) \sim \frac{2^m}{m \log(2)}$.

It has been shown in [8] that the exact number of 31-bit primes ($m = 30$) is 50697537. Therefore in our implementation the support of u_0 will contain all monomials M_i and $\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_0\}$ with probability $>1 - \frac{t\,l}{5 \cdot 10^7}$.

We make one more key observation and claim that $\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_{j-1}\}$ for $1 \leq j \leq l$ with high probability: We have

$$\begin{aligned} u_j &= s_{0j} M_0 + s_{1j} M_1 + \cdots + s_{kj} M_t, \\ u_{j+1} &= s_{0,j+1} M_0 + s_{1,j+1} M_1 + \cdots + s_{k,j+1} M_t. \end{aligned}$$

For a given $j > 0$, if $s_{i,j+1} \neq 0$, but $s_{ij} = 0$ then $M_i \in \text{Supp}(u_{j+1})$ but $M_i \notin \text{Supp}(u_j)$. We consider $\Pr[s_{ij} = 0 \,|\, s_{i,j+1} \neq 0]$. If A is the event that $s_{ij} = 0$ and B is the event that $s_{i,j+1} = 0$ then

$$\Pr[A \,|\, B^c] = \frac{\Pr[A] - \Pr[B] \Pr[A \,|\, B]}{\Pr[B^c]} \leq \frac{\Pr[A]}{\Pr[B^c]}.$$

It follows that

$$\frac{\Pr[A]}{\Pr[B^c]} \leq \frac{l/(\pi(2^{m+1}) - \pi(2^m))}{1 - l/(\pi(2^{m+1}) - \pi(2^m))} = \frac{l}{(\pi(2^{m+1}) - \pi(2^m)) - l}.$$

Hence,

$$\Pr[\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_{j-1}\} \,|\, 1 \leq j \leq l] > 1 - \frac{t\,l}{((\pi(2^{m+1}) - \pi(2^m)) - l}.$$

As an example for $m = 30, l = 5, t = 500$, this probability is >0.99993.

Hardy and Ramanujan [5] proved that for almost all integers, the number of distinct primes dividing a number s is $\omega(s) \approx \log\log(s)$. This theorem was generalized by Erdős-Kac which shows that $\omega(s)$ is essentially normally distributed [3]. By this approximation note that

$$\frac{\Pr[A]}{\Pr[B^c]} \leq \frac{\log\log(s_{ij})/(\pi(2^{m+1}) - \pi(2^m))}{1 - \log\log(s_{i,j+1})/(\pi(2^{m+1}) - \pi(2^m))} = \frac{\log(l\log p)}{(\pi(2^{m+1}) - \pi(2^m)) - \log(l\log p)}.$$

Hence the probability that $\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_{j-1}\}$ is $\gtrsim 1 - t\frac{m\log(lm)}{2^m - m\log(lm)}$. As an example for $m = 30, l = 5, t = 500$, this probability is >0.99995.

What does this mean in the context of multivariate factorization over mod $\mathbb{Z}_{p^l}$ for $l > 1$? It means that the solutions to the multivariate diophantine problems occurring in the lifting process will, with high probability, be a subset of the monomials of the solutions of the previous step and these solutions can be computed simply by solving Vandermonde systems by using a machine prime p and hence by an efficient arithmetic using a sparse MDP solver as described in Algorithm 3.

We sum up the observations made in this section in Theorem 1 below.

Theorem 1. *Let p be a randomly chosen m-bit prime, i.e. $p \in [2^m < p < 2^{m+1}]$. With the notation introduced in this section*

$$\Pr(\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_{j-1}\} \text{ for all } 1 \leq j \leq l) > 1 - \frac{t\,l}{((\pi(2^{m+1}) - \pi(2^m)) - l}.$$

This probability can be approximated by

$$\Pr[\text{Supp}\{u_j\} \subseteq \text{Supp}\{u_{j-1}\} \text{ for all } 1 \leq j \leq l] \gtrsim 1 - \frac{t\,m\log(lm)}{2^m - m\log(lm)}.$$

6 Timing Data

In this section we give some experimental data to verify the effectiveness of the methods described in Sects. 4 and 5. In the tables that follow all timings are in CPU seconds and were obtained on an Intel Core i5–4670 CPU running at 3.40 GHz with 16 GB of RAM. For all Maple timings, we set `kernelopts(numcpus=1);` to restrict Maple to use only one core as otherwise it will do polynomial multiplications and divisions in parallel.

6.1 Iterative vs Direct

In this section, we give some data in Table 1 to compare MTSHL-d with the current approach, i.e. implementing MTSHL so that it solves multi-MDP's using iterative approach as explained in Sect. 4. We include also timings for Wang's algorithm which also uses the iterative approach.

We generated r random polynomials in n variables of total degree d with T terms and coefficients from $[1, 99]$ using Maple's `randpoly` command thus `x1^(d+1)+randpoly([x1,\ldots,xn],degree=d,terms=T,coeffs=rand(1..99))` and multiplied them. Then we factored these polynomials using (i) Wang's algorithm, (ii) MTSHL and (iii) MTSHL-d (our new method explained in Sect. 4). All implementations are in Maple. $tX(tY)$ means that the algorithm factored the polynomial in tX CPU seconds and spent tY CPU seconds solving multiterm MDPs. OOM stands for out of memory. As can be seen from the data, MTSHL is significantly faster than Wang's algorithm and the MDP time in MTSHL-d is less than the MDP time in MTSHL by a factor of $r - 1$ or more.

Table 1. Timings for Wang, MTSHL vs MTSHL-d with $r > 2$.

$r/n/d/T$	Wang (MDP)	MTSHL (MDP)	MTSHL-d (MDP)
3/9/10/30	18.94 (16.00)	2.26 (0.60)	1.36 (0.30)
4/9/15/30	OOM	104.72 (23.23)	90.04 (6.55)
3/9/10/50	251.20 (240.77)	8.87 (2.28)	4.99 (0.71)
3/9/15/100	2302.69 (2235.2)	122.36 (28.58)	99.28 (8.17)
3/11/15/100	OOM	272.78 (42.74)	208.35 (11.51)
3/11/10/100	515.98 (424.76)	189.07 (23.90)	146.80 (6.25)
3/11/20/100	OOM	316.12 (66.7)	256.79 (19.22)

6.2 The $\mathbf{p^L}$ Case

In this section, we give some data in Table 2 to compare the current approach, i.e. implementing MTSHL so that it computes a bound l_B and factors staying in modulo $\mathbb{Z}_{p^{l_B}}$ arithmetic, with the p-adic lifting at the last step approach, i.e. the -staying in $\mathbb{Z}_p$ arithmetic approach-, as explained in this Sect. 5.

We generated 2 random polynomials in n variables of total degree d with T with coefficients in $[0, p^l)$ for $p = 2^{31} - 1$. Then we multiplied the two factors over $\mathbb{Z}$ and then factored the product with MTSHL. Since MTSHL does not know what the actual value of l is, it needs to compute the coefficient bound l_B (using Lemma 14 of [4]) and stays in the $\mathbb{Z}_{p^{l_B}}$ arithmetic. It factored the polynomial in $tX(tY)$ seconds where tY denotes the time spent on solving MDP's. Then we factored the polynomial with MTSHL-d which uses p-adic lifting to recover the integer coefficients as explained in Sect. 5. The timings in column MTSHL-d (MDP) (Lift) are the total time, the time spent in MDP and the time spent doing l lifts. The data in Table 2 shows that doing a p-adic lift is much faster than the previous approach.

Table 2. Timings for MTSHL vs MTSHL-d for large integer coefficients.

$n/d/T_{f_i}$	t_{f_i}	l	l_B	MTSHL (MDP)	MTSHL-d (MDP)	(Lift)	
5/10/300	0.07	2	5	5.866 (5.101)	0.438	(0.132)	(0.241)
5/10/500	0.11	2	5	9.265 (7.937)	1.194	(0.186)	(0.480)
5/10/1000	0.23	2	5	14.448 (12.826)	2.202	(0.264)	(1.332)
5/10/300	0.07	4	9	6.923 (6.104)	1.067	(0.156)	(0.553)
5/10/500	0.11	4	9	10.971 (9.737)	1.854	(0.219)	(1.231)
5/10/1000	0.23	4	9	16.943 (15.183)	3.552	(0.350)	(2.632)
5/10/300	0.07	8	17	8.638 (7.596)	2.553	(0.201)	(2.076)
5/10/500	0.11	8	17	13.118 (11.686)	3.101	(0.280)	(2.396)
5/10/1000	0.23	8	17	19.031 (17.225)	4.905	(0.459)	(4.032)

7 Conclusion

We have shown that when the number of factors to be computed ≥ 2 and for the case where the coefficients of the factors are huge, sparse interpolation techniques can be used to speed up multivariate polynomial factorization. The second author has integrated our code into Maple under a MITACS internship with Dr. Jürgen Gerhard of Maplesoft. The new code will become the default factorization algorithm used by Maple's `factor` command for multivariate polynomials with integer coefficients. The old code will still be accessible as an option.

References

1. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties and Algorithms, 3rd edn. Springer, New York (2007). https://doi.org/10.1007/978-0-387-35651-8
2. Geddes, K.O., Czapor, S.R., Labahn, G.: Algorithms for Computer Algebra. Kluwer, Boston (1992)
3. Erdős, P., Kac, M.: The Gaussian law of errors in the theory of additive number theoretic functions. Am. J. Math. **62**, 738–742 (1940)
4. Gelfond, A.O.: Transcendental and Algebraic Numbers. GITTL, Moscow (1952). English translation by Leo F. Boron, Dover, New York (1960)
5. Hardy, G.H., Ramanujan, S.: The normal number of prime factors of a number n. Q. J. Math. **48**, 76–92 (1917)
6. Kaltofen, E.: Sparse hensel lifting. In: Caviness, B.F. (ed.) EUROCAL 1985. LNCS, vol. 204, pp. 4–17. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-15984-3_230
7. Lang, S.: Diophantine Geometry. Wiley, Hoboken (1962)
8. Law, M.: Computing characteristic polynomials of matrices of structured polynomials, Masters thesis (2017)
9. Lee, M.M.: Factorization of multivariate polynomials. Ph.D. thesis (2013)
10. Monagan, M., Tuncer, B.: Some results on counting roots of polynomials and the Sylvester resultant. In: Proceedings of FPSAC 2016, pp. 887–898. DMTCS (2016)

11. Monagan, M., Tuncer, B.: Using sparse interpolation in hensel lifting. In: Gerdt, V.P., Koepf, W., Seiler, W.M., Vorozhtsov, E.V. (eds.) CASC 2016. LNCS, vol. 9890, pp. 381–400. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45641-6_25
12. Steel, A.: Private communication
13. Wang, P.S.: An improved multivariate polynomial factoring algorithm. Math. Comput. **32**, 1215–1231 (1978)
14. Wang, P.S., Rothschild, L.P.: Factoring multivariate polynomials over the integers. Math. Comput. **29**, 935–950 (1975)
15. Yun, D.Y.Y.: The Hensel lemma in algebraic manipulation. Ph.D. thesis (1974)
16. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979). https://doi.org/10.1007/3-540-09519-5_73
17. Zippel, R.E.: Newton's iteration and the sparse Hensel algorithm. In: Proceedings of SYMSAC 1981, pp. 68–72. ACM (1981)
18. Zippel, R.E.: Interpolating polynomials from their values. J. Symb. Comput. **9**(3), 375–403 (1990)
19. Zippel, R.E.: Effective Polynomial Computation. Kluwer, Boston (1993)