Complexity Issues in Bivariate Polynomial Factorization

A. Bostan, STIX
bostan@stix.polytechnique.fr

G. Lecerf, LAMA
lecerf@math.uvsq.fr

B. Salvy, ALGO
Bruno.Salvy@inria.fr

É. Schost, STIX
schost@stix.polytechnique.fr

B. Wiebelt, STIX
wiebelt@stix.polytechnique.fr

ABSTRACT

Many polynomial factorization algorithms rely on Hensel lifting and factor recombination. For bivariate polynomials we show that lifting the factors up to a precision *linear* in the total degree of the polynomial to be factored is sufficient to deduce the recombination by linear algebra, using *trace recombination*. Then, the total cost of the lifting and the recombination stage is subquadratic in the size of the dense representation of the input polynomial. Lifting is often the practical bottleneck of this method: we propose an algorithm based on a faster multi-moduli computation for univariate polynomials and show that it saves a constant factor compared to the classical multifactor lifting algorithm.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms

Algorithm, Theory

Keywords

Computer algebra, polynomial factorization, Hensel lifting, multi-moduli, Tellegen, transposition principle

1. INTRODUCTION

Many multivariate factoring methods follow Zassenhaus' approach [32, 33]. First, with a low probability of failure the problem is reduced to a bivariate polynomial factorization, by Bertini's irreducibility theorem [34, Ch. 19]. Then, one of the two remaining variables is specialized at random. The resulting univariate polynomial is factored and its factors are lifted up to a high enough precision. The final step lies in recombining the lifted factors to get the factors of the original polynomial. In this text we discuss the complexity aspects of this approach.

ISSAC'04, July 4–7, 2004, Santander, Spain.

1.1 Notation

All along this text R denotes a commutative ring with unity, K a field, $\bar{K}$ its algebraic closure, $R[x,y]$ the ring of polynomials in two variables over R, $R[x]_m$ (resp. $R[x,y]_m$) the module of univariate (resp. bivariate) polynomials of degree at most m. The power series ring over K is denoted $K[[x]]$ and its field of fractions $K((x))$. For any polynomial $G \in R[x,y]$, we denote $\deg(G)$ its total degree and its degree with respect to the variable y is written $\deg_y(G)$. Often $K[x,y]$ is viewed as $K[x][y]$, this is why we denote by G' the first derivative of G with respect to y, while we use G_x to denote derivative with respect to x. The resultant of F and G in $K[x,y]$ with respect to y is denoted $\mathrm{Res}_y(F,G)$.

We use Gantmacher's notation $w_{1:k}$ to denote the k-tuple $(w_1,\ldots,w_k)$. We also use the notation $\langle w_1,\ldots,w_k\rangle = \langle w_{1:k}\rangle$ to denote the K-vector space generated by $w_{1:k}$.

For our complexity analyses, we use the *computation tree* model [6, Ch. 4]. In the case of multiplication of polynomials, we restrict ourselves to straight-line programs performing only *linear* operations in the coefficients of the polynomials (this is the case for the naive, Karatsuba and FFT multiplications). We denote by $\mathsf{M}(n)$ the complexity of multiplying two polynomials of degree at most n in this model. As in [9, §8.3] we assume that M is super-additive: $\mathsf{M}(n_1+n_2) \geq \mathsf{M}(n_1)+\mathsf{M}(n_2)$ for any positive integers n_1 and n_2. For simplicity we also assume that $n\log n \in \mathcal{O}(\mathsf{M}(n))$. We use the notation $f(n) \in \tilde{\mathcal{O}}(g(n))$ to indicate $f \in \mathcal{O}(g(n)(\log n)^a)$ for some $a \geq 0$. The constant ω denotes the matrix multiplication exponent as in [9, Ch. 12], so that two $n \times n$ matrices can be multiplied within $\mathcal{O}(n^\omega)$ field operations ($2 \leq \omega \leq 3$).

1.2 Hypotheses

The polynomial to be factored is always denoted by $F \in K[x,y]$, d represents its total degree. In all this work, we assume the following:

$$\text{Hypothesis (H)} \quad \begin{cases} (i)\ \deg_y(F) = \deg(F) = d; \\ (ii)\ \mathrm{Res}_y(F', F)(0) \neq 0. \end{cases}$$

Hypothesis (H) is not really restrictive: if F is square-free, it can be ensured by means of a generic linear change of variables. Under Hypothesis (H), F is monic in y. The monic (with respect to y) irreducible factors of F over K (resp. over $K[[x]]$) are then denoted $F_{1:r}$ (resp. $\mathfrak{F}_{1:s}$).

1.3 Overview of the Algorithm, Contributions

Suppose that K supports effective univariate polynomial factorization. Then, under Hypothesis (H), we consider the following bivariate factorization algorithm:

1. Fiber. Factor $F(0,y)$, which gives $\mathfrak{F}_{1:s} \bmod x$;

2. Lifting. By Hensel lifting, compute $\mathfrak{F}_{1:s} \bmod x^\sigma$;

3. Recombination. Compute a basis in *reduced echelon form* of the vector space $\pi(L_\sigma)$, where L_σ is the subspace of $K^s \times K[x,y]_{d-1}$ of all the $(\ell_{1:s}, Q)$ satisfying

$$\sum_{i=1}^{s} \ell_i \frac{\mathfrak{F}_i'}{\mathfrak{F}_i} F = Q + \mathcal{O}(x^\sigma), \tag{1}$$

$\pi(L_\sigma)$ denoting the canonical projection of L_σ to K^s.

Stages 1 and 2 are classical in approaches following Zassenhaus; Stage 3 originates in Sasaki's work [26, 24, 25], in a different but equivalent form called *trace recombination* (this equivalence is proved in §2.3). After [1], we call this third stage the *logarithmic derivative recombination* method. The idea is that for σ large enough, any (nonnecessarily irreducible) factor of F corresponds to a solution of (1), as can be seen by considering its logarithmic derivative.

In this text, we do not discuss Stage 1. If K has characteristic zero or large enough, our first contribution is the proof that if $\sigma \geq 3d-2$ then the factors $F_{1:r}$ can be read off the basis in row echelon form of $\pi(L_\sigma)$. A sharper linear bound $\sigma = d+1$ was previously announced in [24]; however, here is a counter-example to this claim:

EXAMPLE 1. *Let* $K = \mathbb{Q}[\alpha]$, *with* $\alpha^2 + \frac{14348907}{2000000} = 0$, *and*

$$F = y^3 + xy^2 + \left(-1 - \frac{4000}{6561}\alpha x + x^2\right) y - \frac{1}{3}x - \frac{32000}{59049}\alpha x^2 + x^3,$$

satisfying $F(0,y) = y(y-1)(y+1)$. *At precision* $\sigma = d+1 = 4$, *one gets* $\pi(L_4) = \langle (1,0,2), (0,1,-1) \rangle$ *which is not sufficient to read the factorization. Then, at precision* $d+2 = 5$, *it appears that* F *is irreducible, since* $\pi(L_5) = \langle (1,1,1) \rangle$.

Thus, to the best of our knowledge, ours is the first linear bound to be proved. Theorem 15.18 in [9] then implies that the Lifting stage requires $\mathcal{O}(\mathsf{M}(d)^2 \log s)$ operations in K.

We study the cost of two Recombination algorithms: a deterministic one requiring $\mathcal{O}(s\mathsf{M}(d)^2 + d^2 s^{\omega-1})$ operations in K and a faster probabilistic one requiring $\mathcal{O}(d^2 + s\mathsf{M}(d) + ds^{\omega-1} + s^\omega \log s)$ operations in K. Once the basis is computed, from $\mathfrak{F}_{1:s}$ at precision at least $\mathcal{O}(x^{d+1})$, recovering $F_{1:r}$ takes $\mathcal{O}(\mathsf{M}(d)^2 \log s)$ operations in K [9, Lemma 10.4].

Roughly speaking, over finite fields of characteristic sufficiently large, this yields a deterministic factorization algorithm of complexity in $\tilde{\mathcal{O}}(d^{\omega+1})$ and a probabilistic Monte Carlo algorithm of complexity in $\tilde{\mathcal{O}}(d^\omega)$ (the dependence in the cardinality of the base field is not taken into account).

These estimates are subquadratic with respect to the size d^2 of the input. This improves Lenstra's $\mathcal{O}(d^8)$ result [20], those of von zur Gathen and Kaltofen [11] and Noro and Yokoyama [23] with complexity $\mathcal{O}(d^6)$ and Gao's $\tilde{\mathcal{O}}(d^4)$ result [7, Additional Remark 3]. Thus, our second contribution is an improvement upon the known polynomial time algorithms for bivariate factorization.

Our last contribution is a new lifting scheme based on a multi-moduli algorithm for univariate polynomials, that generalizes the evaluation algorithm of [4]. Asymptotically, we save a constant factor compared to Shoup's multifactor algorithm described in [9, Ch. 15]. Our C++ implementation on top of NTL [27] allows us to factor the 11th bivariate Swinnerton-Dyer polynomial in $\mathbb{Z}/754974721\mathbb{Z}$ within 30 minutes; the total degree is $d = 2048$ and $s = 1024$, $r = 1$.

1.4 Related Works

We refer to [9, 7, 17, 23, 34] and references therein for details on existing algorithms and the history of polynomial factorization. We only discuss the *trace recombination* technique and related methods.

The trace recombination method was introduced for algebraically closed fields by T. Sasaki *et alii* [26]. In [25], this hypothesis is dropped and the general trace recombination method first appears. Also, perspectives for factoring in $\mathbb{Z}[x]$ are mentioned. In [22] non-monic polynomials are considered. van Hoeij [13] came up with the last missing ingredient for $\mathbb{Z}[x]$: after factoring modulo a prime number p and doing p-adic lifting of the factors, the right recombination is recovered by means of lattice reduction.

The fact that the logarithmic derivative corresponds to the generating series of the traces is well known. But the idea of using it inside the recombination stage is the key to determining accurate bounds on σ. To our best knowledge, this idea was first introduced in a recent work of Belabas, van Hoeij, Klüners and Steel [1]: in the context of an arbitrary global field, they prove that the Hensel and recombination scheme has polynomial complexity. In the bivariate polynomial case they reach the bound $\sigma \geq d(d-1)+1$, whatever the characteristic of the base field is. Their bound follows from the Bézout bound on the degree a certain resultant (occuring in formula (2) below), that turns effective and quantitative the classical argument in Galois theory used in [24, Th. 3.2]. Our method for obtaining a linear bound relies on this quadratic bound: we prove that the quadratic bound implies the linear one if the characteristic of the base field is at least $d(d-1)+1$. Our original contribution consists in differentiating the residues expressing the ℓ_i.

As for *Hensel lifting*, we use the dense model for polynomial representation. Hensel lifting has actually been studied in many other models: sparse polynomials [10, 15], arithmetic circuits [16], black boxes [18], parallel circuits [2]. Subsequent to [32], various algorithms for Hensel lifting in the dense representation have been proposed [30, 21, 33, 29, 14, 8, 2]. The fastest known is the *multifactor lifting* described in [9, Algorithm 15.17], which was first written in C++ by Shoup for his library NTL [27] for factoring in $\mathbb{Z}[x]$. The first fast multi-moduli algorithm appeared in [31, Lemma 4.2], we refer to [6, 9] for historical details.

2. ALGORITHMS AND COMPLEXITY

We consider the family $\mu_{1:r}$ of vectors of $\{0,1\}^s$, defined by $F_i = \prod_{j=1}^{s} \mathfrak{F}_j^{\mu_{i,j}}$. The vectors μ_i can be ordered to form a reduced echelon sequence. By uniqueness of reduced echelon forms, as soon as $\pi(L_\sigma) = \langle \mu_{1:s} \rangle$ holds then the μ_i can be read off from the reduced echelon basis of $\pi(L_\sigma)$.

First, we discuss the required precision σ. Then we estimate the complexity of the algorithm and of an enhanced probabilistic version. Then we show the equivalence between the logarithmic derivative and trace recombination methods. Next, we discuss an optimization that often helps decrease the required precision, and give experimental timings.

2.1 Precision of the Lifting

According to the above notation, $\langle \mu_{1:r} \rangle \subseteq \pi(L_\sigma)$ since:

$$\frac{F_i'}{F_i} F = \sum_{j=1}^{s} \mu_{i,j} \frac{\mathfrak{F}_j'}{\mathfrak{F}_j} F,$$

and $\frac{F_i'}{F_i}F \in K[x,y]_{d-1}$. This inclusion turns out to be an equality as soon as $\sigma \geq 3d-2$:

THEOREM 1. *Under Hypothesis (H), if $\sigma \geq 3d-2$ and if the characteristic of K is either zero or at least $d(d-1)+1$ then $\pi(L_\sigma) = \langle \mu_{1:r} \rangle$.*

PROOF. Let $(\ell_{1:s}, Q) \in L_\sigma$. Let $\varphi_j(x)$ denote any series solution of $\mathfrak{F}_j(x, \varphi_j(x)) = 0$ in $\bar{K}((x))$. The condition on $\mathrm{Res}_y(F', F)(0)$ implies that $\varphi_j \in \bar{K}[[x]]$. Specializing (1) at $y = \varphi_j(x)$ yields

$$Q(x,\varphi_j) = \ell_j \frac{\mathfrak{F}_j' F}{\mathfrak{F}_j}(x,\varphi_j) + \mathcal{O}(x^\sigma) = \ell_j F'(x,\varphi_j) + \mathcal{O}(x^\sigma).$$

Let $\lambda_j = Q(x,\varphi_j)/F'(x,\varphi_j) \in \bar{K}[[x]]$. Then we have $\lambda_j = \ell_j + \mathcal{O}(x^\sigma)$. We are going to show the stronger equality $\lambda_j = \ell_j + \mathcal{O}(x^{d(d-1)+1})$, or equivalently $Q(x,\varphi_j) = \ell_j F'(x,\varphi_j) + \mathcal{O}(x^{d(d-1)+1})$ for all j. This is sufficient to conclude: Let i be such that $F_i(x,\varphi_j) = 0$ and let $j_{1:t}$ be integers such that $F_i = \mathfrak{F}_{j_1} \cdots \mathfrak{F}_{j_t}$ (in particular j belongs to $j_{1:t}$). Using a classical property of resultants (see [9, Exercise 6.12]), in $\bar{K}((x))[y]$, $B(x) = \mathrm{Res}_y(\ell_j F' - Q, F_i)$ is given by

$$B(x) = \prod_{n=1}^{t} \mathrm{Res}_y(\ell_j F' - Q, \mathfrak{F}_{j_n}) = 0 + \mathcal{O}(x^{d(d-1)+1}). \quad (2)$$

Since B is a polynomial of degree at most $d(d-1)$ it follows that B is identically zero, hence $\ell_j F' - Q \in K[x,y]$ is a multiple of F_i and therefore $\ell_j = \ell_{j_1} = \cdots = \ell_{j_t} = Q(x,\varphi_j)/F'(x,\varphi_j)$. This property being established for any j proves that $\ell_{1:s} \in \langle \mu_{1:r} \rangle$.

In order to prove that $\lambda_j = \ell_j + \mathcal{O}(x^{d(d-1)+1})$, we start from $d\lambda_j/dx \in \mathcal{O}(x^{\sigma-1})$, $j = 1, \ldots, s$. A straightforward calculation gives $d\lambda_j/dx = N_j/{F'}^2(x,\varphi_j)$ where

$$\begin{aligned} N_j = {} & (Q_x(x,\varphi_j) + Q'(x,\varphi_j)\varphi_j')F'(x,\varphi_j) \\ & - (F_x'(x,\varphi_j) + F''(x,\varphi_j)\varphi_j')Q(x,\varphi_j) = 0 + \mathcal{O}(x^{\sigma-1}). \end{aligned}$$

Using the fact that $\varphi_j'(x) = -F_x/F'(x,\varphi_j)$, this rewrites $d\lambda_j/dx = A/{F'}^3(x,\varphi_j)$, with

$$A = (Q_x F' - Q' F_x)F' - (F_x' F' - F'' F_x)Q. \quad (3)$$

We now prove that the polynomial A is a multiple of F. This implies that $d\lambda_j/dx = A(x,\varphi_j)/{F'}^3(x,\varphi_j) = 0$ and therefore that $\lambda_j = \ell_j + \mathcal{O}(x^{d(d-1)+1})$, according to the hypothesis on the characteristic of K.

The polynomial A has total degree in x and y at most $3d-4$ and vanishes at the zeroes φ_j of $\mathfrak{F}_j$ for all j, hence at all the zeroes of F in $\bar{K}[[x]][y]$ up to precision $\mathcal{O}(x^{\sigma-1})$. Therefore, so does the remainder R in the Euclidean division of A by F with respect to the variable y. This polynomial obeys $\deg(R) \leq 3d-4$ and $\deg_y(R) \leq d-1$. Letting $R = r_0(x) + r_1(x)y + \cdots + r_{d-1}(x)y^{d-1}$ we have

$$(R(x,\phi_1), \ldots, R(x,\phi_d)) = (r_0, \ldots, r_{d-1})V + \mathcal{O}(x^{\sigma-1}),$$

where $\phi_{1:d}$ are the roots of F in $\bar{K}[[x]][y]$ and V is the transposed $d \times d$ Vandermonde matrix of $\phi_1, \ldots, \phi_d$. Since $\mathrm{Res}_y(F', F)(0) = (\det V)^2(0) \neq 0$, we have that V is invertible over $K[[x]]$ and it follows that $r_{0:d-1} \in \mathcal{O}(x^{\sigma-1})$. From $\deg(r_j) \leq 3d-4 \leq \sigma-2$ we deduce that all the r_j are zero. Thus $R = 0$, which concludes the proof. □

2.2 Recombination

We now present two methods for the Recombination step of the algorithm, leading to different numbers of equations. We recall from [28, Lemma 3.3] that the computation of a reduced echelon solution basis of a linear system with n unknowns and $m \geq n$ equations costs $\mathcal{O}(mn^{\omega-1} + n^\omega \log n)$.

Deterministic Algorithm. In order to compute $\pi(L_\sigma)$, we use the following system with s unknowns and less than σd equations (a similar idea is used in [26]):

$$\begin{aligned} \pi(L_\sigma) = \Big\{ \ell_{1:s} \in K^s \mid {} & \sum_{i=1}^{s} \ell_i \,\mathrm{coeff}\left(\frac{\mathfrak{F}_i'}{\mathfrak{F}_i}F, x^j y^k\right) = 0, \\ & k \leq d-1, d \leq j+k, j \leq \sigma - 1 \Big\}, \end{aligned} \quad (4)$$

where $\mathrm{coeff}\left(G, x^j y^k\right)$ denotes the coefficient of $x^j y^k$ in $G \in K[[x]][[y]]$. The algorithm therefore decomposes into two parts: (i) Compute $\mathfrak{F}_i' F/\mathfrak{F}_i$, for $i = 1, \ldots, s$ and extract the system (4); (ii) find a reduced echelon basis of the solution.

PROPOSITION 1. *Under Hypothesis (H), for $\sigma > 1$, computing the reduced echelon basis of $\pi(L_\sigma)$ from $\mathfrak{F}_{1:s}(0,y)$ requires $\mathcal{O}(s\mathsf{M}(\sigma)\mathsf{M}(d) + \sigma d s^{\omega-1} + s^\omega \log s)$ operations in K.*

PROOF. For $i \leq s$, we first compute $F/\mathfrak{F}_i$ by Euclidean division in $K[[x]][y]$ at precision $\mathcal{O}(x^\sigma)$ (recall that $\mathfrak{F}_i$ is monic in y), then multiply it by $\mathfrak{F}_i'$ to obtain $\mathfrak{F}_i' F/\mathfrak{F}_i$. This requires $\mathcal{O}(\mathsf{M}(\sigma)\mathsf{M}(d))$ operations; summing up over $i \leq s$ this yields $\mathcal{O}(s\mathsf{M}(\sigma)\mathsf{M}(d))$. Extracting the system then has negligible cost $\mathcal{O}(s\sigma d)$. The final term comes from the echelon computation. □

Theorem 1 and [9, Th. 15.18] (or Theorem 2 below) yield:

COROLLARY 1. *Under Hypothesis (H), given $\mathfrak{F}_{1:s}(0,y)$, computing $F_{1:r}$ takes $\mathcal{O}(s\mathsf{M}(d)^2 + d^2 s^{\omega-1})$ operations in K.*

Probabilistic Algorithm. Asymptotically, for s such that $(\mathsf{M}(d)/d)^{2/(\omega-2)} = o(s)$, the most expensive part of the algorithm above is the resolution of the linear system (4) using generic linear algebra methods. A black box method can be used (as in [7]) to obtain a faster (probabilistic) algorithm exploiting the structure of this system.

We now detail a better probabilistic algorithm, by showing that y may be replaced by ax, for a random a in K, with a high probability of success. This leads to a linear system with fewer equations than (4), reducing the cost of the linear algebra stage. This construction is better than only taking random linear combinations of the equations since it avoids computing the whole system. Unfortunately, this requires lifting to a higher precision, which will be denoted by τ instead of σ.

To this end, we first introduce a new linear system that behaves well under specialization of y:

$$\begin{aligned} \Lambda_\tau = \Big\{ \ell_{1:s} \in K^s \mid {} & \sum_{i=1}^{s} \ell_i \,\mathrm{coeff}\left(\frac{\mathfrak{F}_i'}{\mathfrak{F}_i}F, x^j y^k\right) = 0, \\ & k \leq d-1, d \leq j+k \leq \tau - 1 \Big\}. \end{aligned}$$

From (4), it follows that $\pi(L_\tau) \subseteq \Lambda_\tau \subseteq \pi(L_{\tau-d+1})$ and

$$\begin{aligned} \Lambda_\tau = \Big\{ \ell_{1:s} \in K^s \mid {} & \sum_{i=1}^{s} \ell_i \,\mathrm{coeff}\left(\frac{\mathfrak{F}_i' F}{\mathfrak{F}_i}(x, xz), x^j z^k\right) = 0, \\ & k \leq d-1, d \leq j \leq \tau - 1 \Big\}. \end{aligned}$$

Finally, for any $a \in K$ we introduce:

$$\Lambda_\tau^a = \Big\{ \ell_{1:s} \in K^s \mid \sum_{i=1}^{s} \ell_i \operatorname{coeff}\left(\frac{\mathfrak{F}_i' F}{\mathfrak{F}_i}(x, ax), x^j \right) = 0, \quad d \le j \le \tau - 1 \Big\}.$$

LEMMA 1. *For any $b \in K$, there exists $P_b \in K[z]$, such that $P_b \neq 0$, $\deg(P_b) \le (\dim(\Lambda_\tau^b) - \dim(\Lambda_\tau))(d-1)$ and if $P_b(a) \neq 0$ then $\Lambda_\tau = \Lambda_\tau^a \cap \Lambda_\tau^b$.*

PROOF. Obviously we have $\Lambda_\tau \subseteq \Lambda_\tau^a \cap \Lambda_\tau^b \subseteq K^s$, for any a and b. Let $\lambda_{1:\dim(\Lambda_\tau^b)}$ be a basis of Λ_τ^b such that the $\dim(\Lambda_\tau)$ first vectors form a basis of Λ_τ. For any $\lambda_k \notin \Lambda_\tau$ there exists $j \in \{d, \ldots, \tau-1\}$ such that

$$p_k(z) = \sum_{i=1}^{s} \lambda_{k,i} \operatorname{coeff}\left(\frac{\mathfrak{F}_i' F}{\mathfrak{F}_i}(x, zx), x^j \right) \neq 0.$$

We take P_b as the product of all such p_k. By construction each p_k has degree at most $d-1$. □

The probabilistic algorithm now decomposes as: (i) pick a random a and a random b in K; (ii) compute the series expansions $\mathfrak{F}_i' F(x, zx)/\mathfrak{F}_i(x, zx)$ for $z \in \{a, b\}$ and $i = 1, \ldots, s$; (iii) compute the reduced echelon form of $\Lambda_\tau^a \cap \Lambda_\tau^b$.

PROPOSITION 2. *Under Hypothesis (H), from $\mathfrak{F}_{1:s}$ at precision $\mathcal{O}(x^\tau)$, the computation of the reduced echelon basis of $\Lambda_\tau^a \cap \Lambda_\tau^b$ requires $\mathcal{O}(d\tau + s\mathsf{M}(\tau) + \tau s^{\omega-1} + s^\omega \log s)$ operations in K, for any a, b in K and $2\tau \ge s$.*

PROOF. Computing all series $\mathfrak{F}_i(x, ax)$,$\mathfrak{F}_i'(x, ax)$,$F(x, ax)$ at precision $\mathcal{O}(x^\tau)$ takes $\mathcal{O}(d\tau)$ operations. Then, the series $\mathfrak{F}_i' F/\mathfrak{F}_i(x, ax)$ are computed as $\mathfrak{F}_i'(x, ax) \prod_{j \neq i} \mathfrak{F}_j(x, ax)$. To this effect, we compute the products $A_i = \prod_{j<i} \mathfrak{F}_i(x, ax)$, $B_i = \prod_{j>i} \mathfrak{F}_i(x, ax)$, so that $\mathfrak{F}_i'(x, ax) \prod_{j \neq i} \mathfrak{F}_j(x, ax)$ equals $\mathfrak{F}_i'(x, ax) A_i B_i$. Proceeding incrementally, the total cost of this computation is $\mathcal{O}(s\mathsf{M}(\tau))$. The final system has s unknowns and less than 2τ equations. □

Combining the previous results and using $\tau = 4d - 3$, we have $\dim(\Lambda_\tau) = r$ and $\dim(\Lambda_\tau^b) \le s$ and deduce:

COROLLARY 2. *Under Hypothesis (H), for any b in K there exists $P_b \in K[z]$ with $P_b \neq 0$, $\deg(P_b) \le (s-r)(d-1)$, and satisfying: for any $a \in K$ such that $P_b(a) \neq 0$, computing $F_{1:r}$ from $\mathfrak{F}_{1:s}(0, y)$ with the above algorithm requires $\mathcal{O}(\mathsf{M}(d)^2 \log s + d s^{\omega-1} + s^\omega \log s)$ operations in K.*

A bad choice of a and b will result in wrong factors and can be detected by re-expansion. See also §2.4 below.

2.3 Relation to Trace Recombination

We now establish the correspondence between the trace recombination method and the computation of $\pi(L_\sigma)$. Let $P = \prod \mathfrak{F}_i^{\ell_i}$ be a factor of F, and Q a polynomial such that $(\ell_{1:s}, Q) \in L_\sigma$. We use the well known fact that the logarithmic derivative of a polynomial is the generating series of the Newton sums of its roots: in $K[[x]][[y^{-1}]]$,

$$\frac{P'}{P} = y^{-1} \sum_{j \ge 0} y^{-j} \sum_{i=1}^{s} \ell_i \operatorname{Tr}_j(\mathfrak{F}_i), \tag{5}$$

where we denote by $\operatorname{Tr}_j(\mathfrak{F}_i)$ the trace of y^j in the free $K[[x]]$-module $K[[x]][y]/(\mathfrak{F}_i)$.

In view of (1), we have $Q/F = P'/P + \mathcal{O}(x^\sigma)$. This suggests to introduce the *trace recombination* vector space defined for any integers $n \ge 1$ and $\sigma \ge 0$ by

$$\begin{aligned} T_{n,\sigma} = \{ (\ell_{1:s}, p_0, \ldots, p_n) \in K^s \times K[x]_0 \times \cdots \times K[x]_n, \\ p_0 = \textstyle\sum_{i=1}^{s} \ell_i \operatorname{Tr}_0(\mathfrak{F}_i) + \mathcal{O}(x^\sigma), \\ \cdots \\ p_n = \textstyle\sum_{i=1}^{s} \ell_i \operatorname{Tr}_n(\mathfrak{F}_i) + \mathcal{O}(x^\sigma) \}. \end{aligned}$$

We use the same notation π for the canonical projections from $T_{n,\sigma}$ or L_σ to K^s. We now establish that all these projections coincide as soon as $n \ge d-1$, thereby showing the equivalence of our approach and trace recombination.

PROPOSITION 3. *Under Hypothesis (H), for any integer $n \ge d-1$ and any value of σ, $\pi(T_{n,\sigma}) = \pi(L_\sigma)$.*

PROOF. Let $(\ell_{1:s}, Q) \in L_\sigma$. Extracting the coefficients of $y^{-1}, \ldots, y^{-n-1}$ in

$$Q = F \sum_{j=0}^{n} p_j y^{-(j+1)} + \mathcal{O}(y^{-n-2}) \tag{6}$$

defines a family $(p_i)_{i \in \{0,\ldots,n\}}$ with $p_i \in K[x]_i$. From (1),

$$\sum_{j=0}^{n} p_j y^{-(j+1)} = \sum_{j=0}^{n} y^{-(j+1)} \sum_{i=1}^{s} \ell_i \operatorname{Tr}_j(\mathfrak{F}_i) + \mathcal{O}(y^{-n-2}, x^\sigma),$$

which is equivalent to $(\ell_{1:s}, p_{0:n}) \in T_{n,\sigma}$. Now we consider the converse inclusion: let $(\ell_{1:s}, p_{0:n}) \in T_{n,\sigma}$. Since $n \ge d-1$ we define Q by extracting the coefficients of $y^{-1}, \ldots, y^{-d}$ in (6) and check that $Q \in K[x, y]_{d-1}$. Thus, Q rewrites

$$F \sum_{j \ge 0} y^{-(j+1)} \sum_{i=1}^{s} \ell_i \operatorname{Tr}_j(\mathfrak{F}_i) + \mathcal{O}(x^\sigma) = \sum_{i=1}^{s} \ell_i \frac{\mathfrak{F}_i'}{\mathfrak{F}_i} F + \mathcal{O}(x^\sigma),$$

that is $(\ell_{1:s}, Q) \in \pi(L_\sigma)$. □

2.4 Optimization

In most cases, working with precision $\sigma = d+1$ yields the right recombination; building counter-examples such as Example 1 requires some effort. Indeed, by adapting [26, Th. 5.3], we now show that if the reduced echelon form of $\pi(L_\sigma)$ has all its entries in $\{0, 1\}$ for a $\sigma \ge d+1$, then it is not necessary to lift the factors to a higher precision.

PROPOSITION 4. *Under Hypothesis (H), if the characteristic of K is either 0 or at least d and if the reduced echelon form of $\pi(L_\sigma)$ has all its entries in $\{0, 1\}$ for a $\sigma \ge d+1$, then $\pi(L_\sigma) = \langle \mu_{1:r} \rangle$.*

PROOF. Since $\langle \mu_{1:r} \rangle \subseteq \pi(L_\sigma)$, it is sufficient to prove that any vector of $\pi(L_\sigma)$ with all coordinates in $\{0, 1\}$ belongs to $\langle \mu_{1:r} \rangle$, or in other words corresponds to a factor of F over K. We consider such a vector $\ell_{1:s} \in \{0, 1\}^s$ and let $Q \in K[x, y]_{d-1}$ be such that $(\ell_{1:s}, Q) \in L_\sigma$. Let $A = \prod_{i=1}^{s} \mathfrak{F}_i^{\ell_i}$ denote the candidate factor. The proof consists in showing that if $m = \deg_y(A)$, then $A \bmod x^\sigma \in K[x, y]_m$. Then, since $\ell_{1:s} \in \{0, 1\}^s$, we define the cofactor $B = \prod_{i=1}^{s} \mathfrak{F}_i^{1-\ell_i}$ and the same reasoning leads to $B \bmod x^\sigma \in K[x, y]_{d-m}$. It follows from $\sigma \ge d+1$ that $AB = F + \mathcal{O}(x^\sigma)$ implies $(A \bmod x^\sigma)(B \bmod x^\sigma) = F \in K[x, y]$, as was to be proved.

In order to compute the bound on the degree of A, let a_i denote the coefficient of y^i in A and $m = \deg_y(A)$. By construction we have $A'F = QA + \mathcal{O}(x^\sigma)$. Extracting the

Table 1: Deterministic Algorithm in C++

n	d	new lift.	old lift.	eqs	ech	matrix size
6	64	0.42	0.54	0.35	0.04	32×2080
7	128	1.72	4.48	3.19	0.86	64×8256
8	256	8.19	35.1	27.3	13.0	128×32896
9	512	38.2	168	234	198	256×131328
10	1024	177	786	2007	3108	512×524800

Table 2: Probabilistic Algorithm in C++

n	d	fiber	new lift.	eqs	ech	matrix size
6	64	0.03	0.65	0.02	0.01	32×128
7	128	0.13	3.04	0.13	0.02	64×256
8	256	0.37	15.4	0.57	0.18	128×512
9	512	1.11	73.2	2.56	1.55	256×1024
10	1024	3.68	348	11.4	12.1	512×2048
11	2048	13.4	1700	52.2	95.2	1024×4096

coefficients of $y^{d-1}, \ldots, y^{m+d-1}$ gives a triangular system for the a_j's. The hypothesis on the characteristic of K shows that this system defines each a_j as a polynomial in $K[x]_{m-j}$ up to precision $\mathcal{O}(x^\sigma)$. Thus, $A \bmod x^\sigma$ belongs to $K[x,y]_m$. □

For the probabilistic strategy described in §2.2 the bound $d+1$ in this Proposition has to be replaced by $2d$.

2.5 Experiments

We provide timings obtained with our C++ implementation for the Swinnerton-Dyer family $(S_n)_{n\geq 0}$ of bivariate polynomials [34, p. 340] over $\mathbb{Z}/754974721\mathbb{Z}$:

$$S_n(x,y) = \prod (y \pm \sqrt{x+1} \pm \cdots \pm \sqrt{x+n}).$$

S_n has total degree $d = 2^n$, it is monic and for our examples, Hypothesis (H) is satisfied for $x = 7$. These polynomials exhibit the worst possible case for our algorithm, since they are irreducible but their specializations $S_n(7,y)$ split into factors of degree 2, hence $s = 2^{n-1}$.

Proposition 4 always applies in all the given timings, hence the precision is $\sigma = d+1$ for the deterministic approach and $\tau = 2d$ for the probabilistic one. In Table 1 we report timings (in seconds, measured with a 2 GHz Athlon platform) for the deterministic approach and in Table 2 for the probabilistic method. We have implemented two lifting algorithms: our *new lifting* algorithm presented in the next section and *multifactor lifting* as described in [9, Algorithm 15.17]. In Table 1, the columns successively indicate timings for both versions of lifting, for constructing the linear system (eqs) and for the computation of the reduced echelon form (ech). The last column gives the size of the matrix to be reduced. In Table 2, we also indicate the time for the factorization of $F(0,y)$ (fiber), which does not depend on the version of algorithm. This uses NTL's factorization [27].

Our C++ experimental code shows that the theoretical complexities can be observed for high degrees:
– The cost of linear algebra appears to be multiplied by roughly 16 (resp. 8) in the deterministic (resp. probabilistic) algorithm when d is doubled. This is in agreement with our estimates, as we use an implementation where $\omega = 3$.
– The cost of lifting is multiplied by slightly more than 4 when the degree is multiplied by 2. Again, this is in line with the leading term of our complexity estimates and shows that the polynomial multiplication we are using is quite good.
– In the deterministic approach, the cost of linear algebra starts to dominate that of lifting as soon as $n \geq 7$. The advantage of the probabilistic approach then starts to be perceptible. It gains clearly for $n \geq 8$.
– The ratio between both liftings is close to 4.4, which differs from the ratio of the theoretical constants given at the end of the next section. One reason for this is that these analyses give worst-case complexities, whereas the polynomial S_n has all its factors $\mathfrak{F}_i$ of the same degree: in this case, theoretical estimates can be sharpened. A first analysis then leads to a theoretical ratio asymptotic to 13.5/4.

Table 3: Comparison of Implementations

n	Maple 9	Magma 2.9-15	Asir	our code
3	0.589	0.04	0.01	0.02
5	3049	4.36	2.36	0.75
7	$> 150e3$	$92e3$	$> 150e3$	17.7
9	Error	$> 400e3$	Error	394

In Table 3, we compare our code and several computer algebra systems, on the same machine. The family of polynomials for this test is $T_n(x,y) = S_n(x^2,y)S_n(y^2,x)$ that we factor over $\mathbb{Z}/754974721\mathbb{Z}$. For odd n, these polynomials are monic, of degree 2^{n+1}. These polynomials being symmetric, the time to factor them does not depend on the choice of variable in the specialization step.

On these examples, the best timings are provided by Asir 20031213 and Magma 2.9-15 (Magma 2.10-12 gives longer computation times). Singular and Mathematica (both 4.0 and 5.0) could not be compared, as they returned error messages. Maple and Asir return error messages for $n = 9$.

3. HENSEL LIFTING

In actual practice, the lifting stage dominates the process, while the linear algebra part is relatively fast. This motivates a more precise study of the lifting stage, which is the aim of this section. Given F in $K[x,y]$, the question is to lift a factorization of $F(0,y)$ in $K[y]$ to a factorization of F in $K[[x]][y]$. Our solution improves the complexity of the previous algorithms by a constant factor. A basic tool is an improvement of *simultaneous modular reduction*, which we present first. We recall that R is a commutative ring with unity. We define the *reversal* endomorphism $\mathrm{rev}(d,.)$ by $\mathrm{rev}(d,a) = y^d a(1/y)$ for all $a \in R[y]_d$.

3.1 Simultaneous Modular Reduction

Let $p_{0:s}$ be monic polynomials in $R[y]$, of positive degrees $d_{0:s}$; we write $p = p_0 \cdots p_s$ and $d = \deg(p)$. In what follows we improve (by a constant factor) the complexity of the known algorithms for the following problem:

SIMULTANEOUS MODULAR REDUCTION. Given $q \in R_{d-1}[y]$, compute the polynomials $q \bmod p_i$, $0 \leq i \leq s$.

In [4] the case where $\deg(p_i) = 1$ (that is, multipoint evaluation) was treated. The method consisted in designing an algorithm for the *transposed* problem and then transposing it. We now extend this approach to the general problem. For our applications to Hensel lifting, we need an algorithm derived from simultaneous modular reduction, called InverseUpTree and discussed at the end of this section.

The Subproduct Tree. A common piece of our algorithms is the precomputation of the *subproduct tree* T with leaves

$p_{0:s}$ [9, §10.1]. This tree is defined recursively, together with the sequence of integers s_i, by

$$\begin{aligned} T_{0,j} &= p_j, && \text{for } j \in \{0,\dots,s\}, \quad s_0 = s+1,\\ T_{i,j} &= T_{i-1,2j}T_{i-1,2j+1}, && \text{for } j < r_i = \lfloor s_i/2 \rfloor, \qquad i \geq 1. \end{aligned}$$

If $s_i = 2r_i + 1$ we let $T_{i,r_i} = T_{i-1,s_i-1}$ and $s_{i+1} = r_i + 1$, otherwise we just let $s_{i+1} = r_i$. Let $h(T)$ be the smallest integer h such that $s_h = 1$ (the height of the tree). At the top of the tree we get $T_{h(T),0} = p$.

We denote by SubProductTree$(p_0,\dots,p_s)$ the algorithm performing these operations. By [9, Lemma 10.4], its complexity is within $\mathsf{M}(d)\log s + \mathcal{O}(d\log s)$ operations in R.

Linear Combination of Moduli. Let $c_{0:s}$ be in $R[y]$, with $\deg(c_j) < d_j$. The following algorithm [9, Algorithm 10.9] takes as input $c = (c_0,\dots,c_s)$ and outputs the polynomial

$$b = \sum_{j=0}^{s} c_j \frac{p}{p_j}. \tag{7}$$

UpTree(c)
  $b \leftarrow c$;
  for $i \leftarrow 0$ **to** $h(T)-1$ **do**
    for $j \leftarrow 0$ **to** $r_i - 1$ **do**
      $b_j \leftarrow T_{i,2j+1}b_{2j} + T_{i,2j}b_{2j+1}$;
    if $s_i = 2r_i + 1$ **then** $b_{r_i} \leftarrow b_{s_i-1}$;
  return b_0;

This algorithm computes a linear function from $R[y]_{d_0-1} \times \cdots \times R[y]_{d_s-1}$ to $R[y]_{d-1}$; in the sequel, we need algorithms that compute either the transpose or the inverse of this map.

We discuss the transposed version first. We use the transposition techniques of [4], using *transposed multiplication*: given $a \in R[y]$ of degree j, we denote $\mathrm{mul}^t(i,a,.) : R[y]_{i+j} \to R[y]_i$ the transpose of the map $\mathrm{mul}(i,a,.) : R[y]_i \to R[y]_{i+j}$ defined by $\mathrm{mul}(i,a,b) = ab$. The operation mul^t is detailed in [12, 4]; its complexity is $\mathsf{M}(k) + \mathcal{O}(k)$ operations in R, with $k = \max(i,j)$. Using this operation, we deduce the algorithm TUpTree.

TUpTree(b)
  $c_0 \leftarrow b$;
  for $i \leftarrow h(T)-1$ **downto** 0 **do**
    if $s_i = 2r_i + 1$ **then** $c_{s_i-1} \leftarrow c_{r_i}$;
    for $j \leftarrow r_i - 1$ **downto** 0 **do**
      $n_{2j} \leftarrow \deg(T_{i,2j+1}) - 1$;
      $n_{2j+1} \leftarrow \deg(T_{i,2j}) - 1$;
      $c_{2j+1} \leftarrow \mathrm{mul}^t(n_{2j}, T_{i,2j}, c_j)$;
      $c_{2j} \leftarrow \mathrm{mul}^t(n_{2j+1}, T_{i,2j+1}, c_j)$;
  return c;

Such transposition techniques preserve complexity (see [4] and references therein). Thus, once T is computed, using [9, Th. 10.21], it is seen that both UpTree and TUpTree algorithms require $2\mathsf{M}(d)\log s + \mathcal{O}(d\log s)$ operations in R.

Simultaneous Modular Reduction. We first consider the problem obtained by transposing simultaneous modular reduction, viewed as a linear function of q. Given $s+1$ polynomials c_j, with $\deg(c_j) < d_j$, this computes the first d coefficients of the Taylor expansion of

$$\sum_{j=0}^{s} \frac{e_j}{\mathrm{rev}(d_j,p_j)} = \frac{1}{\mathrm{rev}(d,p)} \sum_{j=0}^{s} \frac{e_j \mathrm{rev}(d,p)}{\mathrm{rev}(d_j,p_j)},$$

where $e_j = c_j\mathrm{rev}(d_j,p_j) \bmod y^{d_j}$. For $j \leq s$, let $f_j = \mathrm{rev}(d_j-1, e_j)$; then the right-most sum can be computed as $\mathrm{rev}(d-1, \mathrm{UpTree}(f_0,\dots,f_s))$.

To prove this assertion, we view the simultaneous modular reduction map as the direct product π of the canonical projections $\pi_j : R[y]/(p) \to R[y]/(p_j)$. Therefore, the transpose of π is the sum of the transposes of π_j. Now, any element ℓ_j of the dual of $R[y]/(p_j)$ identifies with its generating series $\sum_{i\geq 0} \ell_j(y^i)y^i$, which is rational and equals $(\ell_j\mathrm{rev}(d_j,p_j) \bmod y^{d_j})/\mathrm{rev}(d_j,p_j)$, see [5, Prop. 1].

We deduce the following transposed modular reduction.

TSimulMod(c)
  $\alpha \leftarrow 1/\mathrm{rev}(d,p) \bmod y^d$;
  for $j \leftarrow 0$ **to** s **do**
    $e_j \leftarrow (c_j\mathrm{rev}(d_j,p_j) \bmod y^{d_j})$;
    $f_j \leftarrow \mathrm{rev}(d_j-1, e_j)$;
  $u \leftarrow \mathrm{UpTree}(f_0,\dots,f_s)$;
  $t \leftarrow \mathrm{rev}(d-1,u)$;
return $\mathrm{mul}(d-1,\alpha,t) \bmod y^d$;

To analyse the complexity of this algorithm, we suppose that the subproduct tree T is already known. Using the super-additivity of M, we get that the **for** loop requires at most $\mathsf{M}(d) + \mathcal{O}(d)$ operations in R. The complexity of computing α is in $\mathcal{O}(\mathsf{M}(d))$ as well, hence the complexity of our algorithm is in $2\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(d))$ operations in R.

Transposing backwards, we obtain the following algorithm for simultaneous modular reduction. Note that for $a \in R[y]$ and $k \geq 0$, the linear maps $R[y]_{k-1} \to R[y]_{k-1}$ defined by $b \mapsto \mathrm{rev}(k-1, ab \bmod y^k)$ and $b \mapsto (a\mathrm{rev}(k-1,b)) \bmod y^k$ are self-adjoint; this explains that some parts are left unchanged by transposition.

SimulMod(q)
  $\alpha \leftarrow 1/\mathrm{rev}(d,p) \bmod y^d$;
  $t \leftarrow \mathrm{rev}(d-1,q)$;
  $u \leftarrow \mathrm{mul}(d-1,\alpha,t) \bmod y^d$;
  $f \leftarrow \mathrm{TUpTree}(u)$;
  for $j \leftarrow 0$ **to** s **do**
    $e_j \leftarrow (f_j\mathrm{rev}(d_j,p_j) \bmod y^{d_j})$;
    $c_j \leftarrow \mathrm{rev}(d_j-1, e_j)$;
  return $(c_0,\dots,c_s)$;

Again, the complexity equals $2\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(d))$, if T is already known. Taking the cost of computing T into account yields a cost of $3\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(d))$ operations in R. This is to be compared with the complexity of [9, Cor. 10.17], which was in $11\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(d))$.

Inverse UpTree. We conclude this subsection by giving an algorithm to invert the UpTree map. The input is a polynomial $b \in R[y]_{d-1}$ and the output is a vector $(c_0,\dots,c_s) \in R[y]_{d_0-1} \times \cdots \times R[y]_{d_s-1}$ such that (7) holds. For the output to be uniquely defined, we make the additional assumption (satisfied in what follows) that p' is invertible modulo p. In this case, it is then noted in [3, § 23] that the coefficients c_j are given by the interpolation formula

$$b = \sum_{j=0}^{s} c_j \frac{p}{p_j} = \sum_{j=0}^{s} \left(b\frac{p_j'}{p'} \bmod p_j \right) \frac{p}{p_j}.$$

This leads to the following algorithm:

InverseUpTree(b)
$r \leftarrow b/p' \bmod p$;
$(u_0, \ldots, u_s) \leftarrow \text{SimulMod}(r)$;
for $j \leftarrow 0$ **to** s **do**
$c_j \leftarrow u_j p'_j \bmod p_j$;
return $(c_0, \ldots, c_s)$;

To state the complexity of this algorithm, we denote by $\mathsf{Inv}(d)$ the complexity of computing $1/p' \bmod p$. If the Extended Euclidean Algorithm can be applied in R, by [9, Th. 11.7], $\mathsf{Inv}(d) = \mathcal{O}(\mathsf{M}(d)\log d)$; in our application to Hensel lifting, other techniques will be used.

The cost for simultaneous reduction was given above. To conclude the analysis, we note that each polynomial c_j can be computed in $6\mathsf{M}(d_j)+\mathcal{O}(d_j)$ operations in R [9, Cor. 9.7], so using the super-additivity of M, the total cost of the **for** loop is within $\mathcal{O}(\mathsf{M}(d))$ operations in R. Assuming that T is precomputed, the number of operations in R used by the algorithm InverseUpTree is $\mathsf{Inv}(d) + 2\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(d))$.

3.2 Lifting Algorithm

We now address the main issue of this section, Hensel lifting. Let $F \in K[x,y]$ satisfy hypothesis (H) and let $\mathfrak{F}_{1:s}$ be its irreducible factors in $K[[x]][y]$. Given $\mathfrak{F}_i \bmod x$, our goal is to compute the polynomials $\mathfrak{F}_i \bmod x^\sigma$.

Hensel lifting proceeds by computing these polynomials at successive precisions $1, 2, \ldots, 2^s, \ldots$; the basic ingredient is thus an algorithm to perform a single lifting step. Let then κ be in $\mathbb{N}$, and write $\mathfrak{f}_i = \mathfrak{F}_i \mod x^\kappa$ and $\mathfrak{f}_i^* = \mathfrak{F}_i \mod x^{2\kappa}$.

For $i \le s$, we write $\mathfrak{f}_i^* = \mathfrak{f}_i + x^\kappa \delta_i$, where δ_i has degree less than κ in x; recall that F and the $\mathfrak{F}_i$'s are monic, so $\deg_y(\delta_i)$ is less than $\deg_y(\mathfrak{F}_i)$. The problem is now reduced to the computation of these polynomials δ_i. To do so, we expand the product $F(x,y)$ as

$$F(x,y) - \prod_{i\le s} \mathfrak{f}_i = x^\kappa \sum_{i\le s} \delta_i \prod_{j\ne i} \mathfrak{f}_j \mod x^{2\kappa}.$$

The left-hand side has valuation x^κ, since $\mathfrak{f}_{1:s}$ forms a factorization of F modulo x^κ. Dividing out x^κ, we obtain

$$\Delta = \sum_{i\le s} \delta_i \prod_{j\ne i} \mathfrak{f}_j \mod x^\kappa, \tag{8}$$

where Δ is $F(x,y) - \prod_{i\le s} \mathfrak{f}_i$, taken modulo $x^{2\kappa}$, and divided by x^κ. Thus, $\deg_x(\Delta) < \kappa$ and $\deg_y(\Delta) < d$. Note that the polynomial Δ is also considered in [29, 14, 8, 2].

To obtain Δ, we compute the subproduct tree T associated to $\mathfrak{f}_1, \ldots, \mathfrak{f}_s$, with coefficients taken modulo $x^{2\kappa}$; this gives $\prod_{i\le s} \mathfrak{f}_i$ modulo $x^{2\kappa}$, from which Δ is deduced. The polynomials δ_i are then obtained by applying algorithm InverseUpTree to Δ with coefficients taken modulo x^κ (the required invertibility assumption is satisfied). The subproduct tree necessary to apply InverseUpTree is obtained by truncating T modulo x^κ. The algorithm is described below:

HenselStep$(\mathfrak{f}_1, \ldots, \mathfrak{f}_s)$
$T \leftarrow \text{SubProductTree}(\mathfrak{f}_1, \ldots, \mathfrak{f}_s) \bmod x^{2\kappa}$;
$\Delta \leftarrow (F - T_{d,0} \mod x^{2\kappa})/x^\kappa$;
$(\delta_1, \ldots, \delta_s) \leftarrow \text{InverseUpTree}(\Delta)$;
return $(\mathfrak{f}_1 + x^\kappa\delta_1, \ldots, \mathfrak{f}_s + x^\kappa\delta_s)$;

To analyze the complexity of this algorithm, recall that the computation of T is performed with coefficients modulo $x^{2\kappa}$. From the results of the previous subsection, this requires $\mathsf{M}(2\kappa)\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(2\kappa)d\log s)$ operations in K.

Then, all computations in InverseUpTree are performed modulo x^κ. Recall that this algorithm requires to compute $1/F' \bmod F$. If $\kappa = 1$, this is done using the Extended Euclidean Algorithm, whose cost is $\mathcal{O}(\mathsf{M}(d)\log d)$ operations in K, so the total cost of HenselStep fits in $\mathcal{O}(\mathsf{M}(d)\log d)$ operations in K. If $\kappa > 1$, we assume that $1/F' \bmod F$ has already been computed modulo $x^{\kappa/2}$ and use a Newton iteration to lift this inverse modulo x^κ. From [9, Th. 9.2], this uses 2 multiplications and 2 additions in $K[x,y]/(F, x^\kappa)$, hence the cost of the inverse computation is in $\mathcal{O}(\mathsf{M}(\kappa)\mathsf{M}(d))$ operations in K. By the results of the previous subsection, we conclude that HenselStep uses $2\,\mathsf{M}(2\kappa)\mathsf{M}(d)\log s + \mathcal{O}(\mathsf{M}(\kappa)\mathsf{M}(d))$ operations in K. Summing up these contributions for $\kappa = 1, 2, \ldots$ yields the complexity result.

THEOREM 2. *Let $\sigma \in \mathbb{N}$ and $\ell = \lceil \log_2 \sigma \rceil$. Given the polynomials $\mathfrak{F}_{1:s} \bmod x$, one can compute $\mathfrak{F}_{1:s} \bmod x^\sigma$ in*

$$4\mathsf{M}(2^\ell)\mathsf{M}(d)\log s + \mathcal{O}\big(\mathsf{M}(d)(\mathsf{M}(2^\ell) + \log d)\big)$$

operations in K.

We conclude this section by comparing our algorithm to the multifactor lifting algorithm [9, Algorithm 15.17]. We first recall the basics of that algorithm; it is enough to describe the inductive step, lifting from precision κ to precision 2κ.

The multifactor lifting algorithm relies on both the subproduct tree T associated to $\mathfrak{F}_1, \ldots, \mathfrak{F}_s$ and a tree of cofactors $\overline{T}$. As a tree, the structure of $\overline{T}$ is the same as that of T, its content being defined by:

$$\overline{T}_{i,2j} = \frac{1}{T_{i,2j+1}} \bmod T_{i,2j}, \quad \overline{T}_{i,2j+1} = \frac{1}{T_{i,2j}} \bmod T_{i,2j+1}$$

with $j < r_i$ and $i < h(T)$, with the notation of the previous subsection. The lifting step then goes as follows: suppose that T is known at precision κ and $\overline{T}$ at precision $\kappa/2$; then lift $\overline{T}$ to precision κ and use it to lift T to precision 2κ.

Both this algorithm and the one we propose share the same tree structure; their overall complexities are in the same asymptotic class, but the constants in the $\mathcal{O}(\ )$ estimates differ. Theorem 2 gives the complexity of our algorithm; the following refines that of [9, Th. 15.8].

THEOREM 3. *Let σ in $\mathbb{N}$ and $\ell = \lceil \log_2 \sigma \rceil$. Given $\mathfrak{F}_{1:s} \bmod x$, the cost of [9, Algorithm 15.17] to $\mathfrak{F}_{1:s}$ mod x^σ is (in terms of operations in K):*

$$13.5\ \mathsf{M}(2^\ell)\mathsf{M}(d)\log s + \mathcal{O}\Big(\mathsf{M}(d)(\mathsf{M}(2^\ell) + \log d)\Big)$$

The detailed proof of the above theorem is omitted here; we only underline the main differences between the two approaches. We also mention that a more precise study of the steps in the multifactor algorithm shows that some of them can be shared; the best we could do decreases the leading term of the complexity to $9\mathsf{M}(2^\ell)\mathsf{M}(d)\log s$.

Lifting $\overline{T}$ to precision κ amounts to lifting all corresponding Bézout identities to this precision. By contrast, our algorithm does not require to handle a whole tree of cofactors; only the inverse of $F' \bmod F$ is required, whose lifting is cheaper by a factor $\log s$. Next, lifting T to precision κ using [9, Algorithm 15.10] amounts to applying the classical bivariate Hensel lifting algorithm to all internal nodes of T, and thus involves Euclidean divisions at each node. Our algorithm only performs Euclidean divisions at the leaves of T; the only operations that are done at internal nodes are transposed multiplications, which are cheaper.

4. CONCLUSION

Requiring the input polynomial to be monic is not restrictive in theory; however, in future work, we expect to generalize the present results in order to avoid changing the coordinates, possibly using Newton polygons as in [22].

Most Chinese remaindering algorithms use simultaneous modular reduction [6, 9, 3]. We also plan to explore this question, using the algorithms of §3.

Finally, our interest in factorization came from the problem of decomposing algebraic varieties in irreducible components. For most base fields met in practice, the results of this paper imply that this operation has the same complexity as the equidimensional decomposition algorithm of [19].

Acknowledgments. We thank K. Belabas, M. van Hoeij, J. Klüners and A. Steel for exchanging preprints and discussing with us during the preparation of this work. We are especially grateful to M. van Hoeij for pointing out a missing hypothesis for Theorem 1 in the first version of this text. We also thank G. Chèze and A. Galligo for fruitful discussions. All calculations were done on the computers of the MEDICIS center, École polytechnique, France. This work was partially supported by *ACI MathSTIC n° 24, 2002*.

STIX, FRE CNRS 2341, École polytechnique, 91128 Palaiseau, France

LAMA, UMR CNRS 8100, Université de Versailles St-Quentin-en-Yvelines, 45 avenue des États-Unis, 78035 Versailles, France

ALGO, Algorithms Project, INRIA Rocquencourt, 78153 Le Chesnay, France

5. REFERENCES

[1] K. Belabas, M. van Hoeij, J. Klüners, and A. Steel. Factoring polynomials over global fields. 2003.

[2] L. Bernardin. On bivariate Hensel lifting and its parallelization. In *ISSAC'98*, pages 96–100. ACM Press, 1998.

[3] D. J. Bernstein. Fast multiplication and its applications. Manuscript, 2003.

[4] A. Bostan, G. Lecerf, and É. Schost. Tellegen's principle into practice. In *ISSAC'03*, pages 37–44. ACM Press, 2003.

[5] A. Bostan, B. Salvy, and É. Schost. Fast algorithms for zero-dimensional polynomial systems using duality. *AAECC*, 14(4):239–272, 2003.

[6] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*. Springer, Berlin, 1997.

[7] S. Gao. Factoring multivariate polynomials via partial differential equations. *Math. Comp.*, 72:801–822, 2003.

[8] J. von zur Gathen. Hensel and Newton methods in valuation rings. *Math. Comp.*, 42(166):637–661, 1984.

[9] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 1st edition, 1999.

[10] J. von zur Gathen and E. Kaltofen. Factoring sparse multivariate polynomials. *J. Comput. System Sci.*, 31:265–287, 1985.

[11] J. von zur Gathen and E. Kaltofen. Factorization of multivariate polynomials over finite fields. *Math. Comp.*, 45:251–261, 1985.

[12] G. Hanrot, M. Quercia, and P. Zimmermann. The Middle Product Algorithm, I. *Appl. Algebra Engrg. Comm. Comput.*, 14(6):415–438, 2004.

[13] M. van Hoeij. Factoring polynomials and the knapsack problem. *J. Number Theory*, 95(2):167–189, 2002.

[14] E. Kaltofen. Polynomial factorization. In B. Buchberger, G. Collins, and R. Loos, editors, *Computer algebra*, pages 95–113. Springer, 1982.

[15] E. Kaltofen. Sparse Hensel lifting. In *EUROCAL'85, Vol. 2 (Linz, 1985)*, volume 204 of *LNCS*, pages 4–17. Springer, Berlin, 1985.

[16] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI, 1989.

[17] E. Kaltofen. Polynomial factorization: a success story. In *ISSAC'03*, pages 3–4. ACM Press, 2003.

[18] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: greatest common divisors, factorization, separations of numerators and denominators. *J. Symb. Comput.*, 9(3):301–320, 1990.

[19] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, 19(4):564–596, 2003.

[20] A. K. Lenstra. Factoring multivariate polynomials over finite fields. *J. Comput. System Sci.*, 2:235–248, 1985.

[21] D. R. Musser. Multivariate polynomial factorization. *J. Assoc. Comput. Mach.*, 22:291–308, 1975.

[22] K. Nagasaka and T. Sasaki. Approximate multivariate polynomial factorization and its time complexity, 1998. preprint.

[23] M. Noro and K. Yokoyama. Yet another practical implementation of polynomial factorization over finite fields. In *ISSAC'02*, pages 200–206. ACM Press, 2002.

[24] T. Sasaki, T. Saito, and T. Hilano. Analysis of approximate factorization algorithm. I. *Japan J. Indust. Appl. Math.*, 9(3):351–368, 1992.

[25] T. Sasaki and M. Sasaki. A unified method for multivariate polynomial factorizations. *Japan J. Indust. Appl. Math.*, 10(1):21–39, 1993.

[26] T. Sasaki, M. Suzuki, M. Kolář, and M. Sasaki. Approximate factorization of multivariate polynomials and absolute irreducibility testing. *Japan J. Indust. Appl. Math.*, 8(3):357–375, 1991.

[27] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net`.

[28] A. Storjohann. *Algorithms for matrix canonical forms*. PhD thesis, ETH, Zürich, 2000.

[29] P. S. Wang. An improved multivariate polynomial factoring algorithm. *Math. Comp.*, 32(144):1215–1231, 1978.

[30] P. S. Wang and L. P. Rothschild. Factoring multivariate polynomials over the integers. *Math. Comp.*, 29:935–950, 1975.

[31] D. Y. Y. Yun. Uniform bounds for a class of algebraic mappings. *SIAM J. on Computing*, 8:348–356, 1979.

[32] H. Zassenhaus. On Hensel factorization I. *J. Number Theory*, 1(1):291–311, 1969.

[33] H. Zassenhaus. A remark on the Hensel factorization method. *Math. Comp.*, 32(141):287–292, 1978.

[34] R. Zippel. *Effective Polynomial Computation*. Kluwer Academic Publishers, 1993.