J. Symbolic Computation (1996) **21**, 377–396

Modular Algorithm for Sparse Multivariate Polynomial Interpolation and its Parallel Implementation

HIROKAZU MURAO[†§¶] AND TETSURO FUJISE[‡‖**]

[†] *Computer Centre, The University of Tokyo, Yayoi 2–11–16, Bunkyo-ku, Tokyo 113, Japan*
[‡] *Institute for New Generation Computer Technology*
Mita Kokusai Bldg. 21F, 4–28, Mita 1–chome, Minato-ku, Tokyo 108, Japan

(Received 31 May 1995)

A new algorithm for sparse multivariate polynomial interpolation is presented. It is a multi-modular extension of the Ben-Or and Tiwari algorithm, and is designed to be a practical method to construct symbolic formulas from numeric data produced by vector or massively-parallel processors. The main idea in our algorithm comes from the well-known technique for primality test based on Fermat's theorem, and is the application of the generalized Chinese remainder theorem to the monomial exponents. We regard the exponent vector of each multivariate monomial as a mixed-radix representation of the corresponding exponent value obtained after the transformation by Kronecker's technique. It is shown by complexity comparison and experimental results that the step for univariate polynomial factorization is most expensive in our algorithm, and its parallelization is considered. Also reported are some empirical results of the parallelization on KLIC, a portable system of a concurrent logic programming language KL1.

1. Introduction

Polynomial interpolation is a simple but important algebraic tool to determine a polynomial formula from a sequence of numeric data. If the objective (unknown) polynomial, given as a black-box which allows evaluations at arbitrary points, is over the integers $\mathbb{Z}$, the numeric data must be integers of arbitrary precisions. According to the well-known Chinese remainder theorem, numeric data required can be constructed from integers of fixed magnitudes, i.e., remainders by fixed moduli. Thus, we can determine a polynomial over $\mathbb{Z}$ only from sufficiently many integer values of fixed-magnitude, and if the moduli are chosen so as to fit in a machine word size, the required numeric data can be processed and generated directly by hardware. This computing method is very effective in calculations

§ Supported in part by Grant-in-Aid for General Scientific Research from the Ministry of Education, Science, Sports and Culture, Japan under Grant (C) 05680266 and Grant (C) 07680337.
¶ E-mail: murao@cc.u-tokyo.ac.jp
‖ Current address: Mitsubishi Research Institute, Otemachi 2–3–6, Chiyoda-ku, Tokyo 100, Japan.
** E-mail: fujise@mri.co.jp

0747–7171/96/040377 + 20 $18.00/0

with intermediate expression growth such as determinant expansions or inversions of matrices with polynomial entries. The method was first invented by Takahasi and Ishibashi (1960), and later applied by one of the authors (Murao, 1991) to a large-scale symbolic formula calculation by using vector processors. Current high-performance computers can execute and generate a large number of required numeric data very quickly. To continue this approach and to make full use of high-performance computers by data-parallel processing, we need a modular algorithm for sparse multivariate polynomial interpolation. That algorithm is requested to work with multiple moduli, in order to expand parallelism in numeric evaluations and in the algorithm itself. In general, multi-modular computations are easy to parallelize (Wang 1990; Villard, 1989; Wang, 1992).

A milestone algorithm for sparse multivariate polynomial interpolation was published by Ben-Or and Tiwari (1988). It was the first deterministic algorithm that accounts for sparsity, while Zippel has presented a probabilistic algorithm in Zippel, (1979) and its deterministic version in Zippel (1990). The Ben-Or and Tiwari algorithm consists of the following two independent stages:

(1) exact determination of monomials existent in a target polynomial, and
(2) determination of their coefficients in the polynomial.

These determinations are based on decoding BCH codes and special substitutions used in Grigoriev and Karpinski (1987); monomials are determined through the factorization over $\mathbb{Z}$ of a feedback connection polynomial for a value sequence, and the coefficients are determined by solving a transposed Vandermonde system.

In this paper, we apply a multimodular technique to the Ben-Or and Tiwari algorithm. The Vandermonde system in the second stage for coefficient determination can be solved efficiently using the algorithm developed by Zippel, and its modular extension is straightforward [for the details, see Zippel (1990); Kaltofen and Lakshman (1988)]. However, the first stage (monomial determination) does not allow direct application of the Chinese remainder theorem, because it is difficult to recover the factorization of a polynomial over $\mathbb{Z}$ from its multiple modular images (Loos, 1983). In order to overcome this difficulty, we shall relax our requirements, on the assumption that the polynomial to be interpolated may be evaluated cheaply, using a massively parallel or vector processor, so that *a set of monomials determined in the first stage should simply give candidates and may include monomials with zero coefficients in the target polynomial.* Reduction of the number of monomial candidates is the key to our approach.

First, under a given degree bound for each variable, we apply Kronecker's technique (Knuth, 1981) to make a given black-box polynomial univariate, and limit our concern to the exponents of monomials in the univariate polynomial. Second, we regard the exponent vector of each multivariate monomial as a mixed-radix representation of the exponent of the corresponding univariate monomial. Evaluations are done using a primitive $(p-1)$st root ξ of unity in $\mathbb{Z}_p$ for multiple primes p. Then, the univariate exponents can be obtained as discrete logarithms with base ξ, and are modulo $(p-1)$. The most significant feature of our algorithm is the application of the generalized Chinese remainder theorem to the exponents†. Note that $p-1$ is often highly composite, and according to the theorem, an arbitrary combination of those logarithms with multiple moduli may not represent

† To the extent of the authors' knowledge, our algorithm will be the first that develops such a use of the theorem.

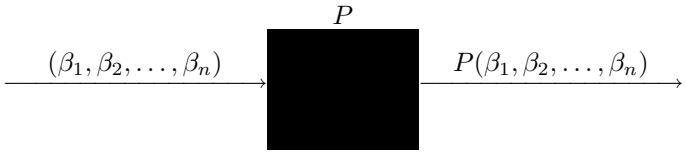
any integer. This imposes a tight constraint on making combinations of modular linear factors of feedback connection polynomials, and resolves the above mentioned problem, reducing the number of possible combinations. Then, every combination of logarithms is converted to a mixed-radix representation to obtain an exponent vector.

Also investigated in this paper is a parallel implementation of the algorithm. We discuss techniques for parallelization and show practical methods used in our implementation. Timing results indicate that parallelization is effective in almost all cases and is most effective for the worst cases of the probabilistic process in polynomial factorization.

In the following sections, we start by describing the basic ideas and algorithms, leading up to a complete description of the new algorithm in Section 4. Section 5 presents a simple example to facilitate an understanding of the algorithm. Section 6 briefly discusses the complexity of the algorithm and methods for its parallelization. Also, the timing data from our empirical studies are given in this section, to demonstrate the performance and to support the discussion. In Section 7, we conclude with some remarks.

1.1. NOTATION

Let $P(x_1, x_2, \ldots, x_n)$ be an n-variate unknown polynomial over $\mathbb{Z}$ to be determined, and given as a black-box which allows evaluations at arbitrary points $\in \mathbb{Z}^n$.



The problem of polynomial interpolation is the exact determination of the polynomial representation of P. Assume P consists of t distinct monomials such as

$$P(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{t} c_i m_i$$

where $m_i = x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_n^{e_{in}}$ are distinct monomials and c_i are the corresponding non-zero coefficients $\in \mathbb{Z}$. Here, t, c_i and m_i are unknown. Instead, assume that we are given a bound τ for t ($t \leq \tau$), a coefficient bound C ($|c_i| \leq C$), and, for each variable x_j, a degree bound $\bar{d}_j$ of P in x_j ($e_{ij} < \bar{d}_j$ for $1 \leq {}^{\forall}i \leq t$). Let $d = \max_{1 \leq j \leq n}\{\bar{d}_j\}$.

Let d_j be the smallest prime† such that $\bar{d}_j \leq d_j$. Let D_j ($j = 1, 2, \ldots, n+1$) be their product such that

$$D_j = \begin{cases} 1, & j = 1, \\ \prod_{k=1}^{j-1} d_k, & j \geq 2, \end{cases}$$

and $\vec{D}$ denote $(D_1, D_2, \ldots, D_n)$. For a monomial m_i, we define its exponent vector $\vec{e}i = e_{i1}, e_{i2}, \ldots, e_{in}$, and regard it as a mixed-radix representation (Knuth, 1981) of an integer E_i with $\vec{d} = (d_1, d_2, \ldots, d_{n-1})$ such that

$$E_i = e_{i1} + d_1(e_{i2} + d_2(\cdots(e_{in-1} + d_{n-1}\, e_{in})\cdots))$$

† Not necessarily a prime, but must be relatively prime with $(p_k - 1)$, where p_k are later-chosen primes as moduli. In the case of the algorithm (for moderate-sized D_{n+1}) in Section 4.1, this condition is not required and d_j can be $\bar{d}_j$.

$$= D_1 e_{i1} + D_2 e_{i2} + \cdots + D_n e_{in} = \vec{D} \cdot \vec{e}i.$$

This value can be regarded as the degree of each univariate term obtained after the transformation by Kronecker's technique (van der Waerden, 1940): $P(X^{D_1}, X^{D_2}, \ldots, X^{D_n})$. For each E_i with a vector $\vec{G} = (G_1, G_2, \ldots, G_s) \in \mathbb{Z}^s$, we denote $\varepsilon_{ij} = E_i \bmod G_j$, and its vector $\vec{\varepsilon}i = (\varepsilon_{i1}, \varepsilon_{i2}, \ldots, \varepsilon_{is})$.

All classical polynomial interpolation algorithms assume the occurrence in P of every possible combination of $\vec{e}_i$, i.e., all the integral values 0 through $D_{n+1} - 1$ of E_i. However, for a modern sparse polynomial interpolation algorithm, it is desired to determine as small a set $\{\vec{e}i\}$ for P as possible. This can be done via an auxiliary polynomial for a value sequence of P as described in Section 2. For a sequence a_i, $i = 0, 1, \ldots, N$, we call

$$\Lambda(z) = z^l + \lambda_{l-1} z^{l-1} + \cdots + \lambda_1 z + \lambda_0$$

a *feedback connection polynomial* of the sequence, if its coefficients satisfy the following linear relation (Blahut, 1985):

$$\begin{pmatrix} a_{N-l} & a_{N-l+1} & \cdots & a_{N-1} \\ a_{N-l-1} & a_{N-l} & \cdots & a_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_0 & a_1 & \cdots & a_{l-1} \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{l-1} \end{pmatrix} = - \begin{pmatrix} a_N \\ a_{N-1} \\ \vdots \\ a_l \end{pmatrix}. \tag{1.1}$$

2. Backgrounds and Outline of The Algorithms

2.1. BASIC MATHEMATICAL FACTS

We consider the value sequence $a_k = P({\rho_1}^k, {\rho_2}^k, \ldots, {\rho_n}^k)$, $k = 0, 1, \ldots, 2t-1$, and denote $b_i = {\rho_1}^{e_{i1}} {\rho_2}^{e_{i2}} \cdots {\rho_n}^{e_{in}}$: $a_k = \sum_i c_i {b_i}^k$. Consider the following $l \times l$ Toeplitz-like matrices:

$$A_l = \begin{pmatrix} a_{l-1} & \cdots & a_{2l-2} \\ \vdots & \ddots & \vdots \\ a_0 & \cdots & a_{l-1} \end{pmatrix}.$$

As pointed out in Ben-Or and Tirwari (1988) and Kaltofen and Lakshman (1988), A_t can be factorized into

$$A_t = B \begin{pmatrix} c_1 & 0 & \ldots & 0 \\ 0 & c_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & c_t \end{pmatrix} B^{\mathrm{T}} \qquad \text{where } B = \begin{pmatrix} {b_1}^{t-1} & {b_2}^{t-1} & \ldots & {b_t}^{t-1} \\ {b_1}^{t-2} & {b_2}^{t-2} & \ldots & {b_t}^{t-2} \\ \vdots & \vdots & \vdots\vdots\vdots & \vdots \\ 1 & 1 & \ldots & 1 \end{pmatrix}.$$

Because $\det B = \prod_{1 \le i < j \le t} (b_i - b_j)$ as is well-known, A_t is non-singular if the values b_i are distinct from each other.

LEMMA 2.1. *Let $a_k = \sum_{i=1}^t c_i {b_i}^k$. Assume all b_i are distinct from each other. Then, the polynomial*

$$\Lambda(z) = \prod_{i=1}^{t} (z - b_i) \tag{2.1}$$

is a unique feedback connection polynomial of smallest degree for the sequence a_k, $k = 0, 1, \ldots, N$, *where* $N \geq 2t - 1$.

PROOF. From the condition imposed on b_i, the matrix A_t is non-singular. The degree l of any feedback connection polynomial for the sequence must be $\geq t$, because otherwise the first l columns of A_t are linearly dependent, which contradicts the non-singularity. It is easy to see that the coefficients of the above $\Lambda(z)$ satisfy the linear relation (1.1) with $l = t$ and $N \geq 2t - 1$. The uniqueness follows again from the non-singularity of A_t. □

Note that over $\mathbb{Z}$, if ρ_i are distinct primes, the condition for b_i is satisfied for arbitrary e_{ij}'s, and the prime factorization of b_i's, obtained after the factorization (2.1), gives the exact set of exponent vectors $\vec{e}i$ existent in P.

2.2. THE BEN-OR AND TIWARI ALGORITHM

The Ben-Or and Tiwari algorithm determines the polynomial representation of P from its values at 2τ distinct points $\in \mathbb{Z}^n$ in polynomial-time deterministically (Ben-Or and Tiwari, 1988). The key idea with this algorithm is the uses of the special evaluation points due to Grigoriev and Karpinski (1987) and of the feedback connection polynomial, characterized by (2.1). The following gives an outline of the algorithm.

(0) Letting ρ_i be the ith prime $\in \mathbb{Z}$, evaluate and let

$$a_k \Leftarrow P({\rho_1}^k, {\rho_2}^k, \ldots, {\rho_n}^k) \qquad \text{for } k = 0, 1, \ldots, 2\tau - 1.$$

(1) Find the rank, equal to t, of A_τ, and solve the non-singular Toeplitz system

$$A_t(\lambda_0, \lambda_1, \ldots, \lambda_{t-1})^{\mathrm{T}} = -(a_{2t-1}, a_{2t-2}, \ldots, a_t)^{\mathrm{T}}$$

to obtain λ_i.

(2) Find all the integer roots of $\Lambda(z) = \sum_{i=0}^{t-1} \lambda_i z^i \big(= \prod_{i=1}^{t}(z - b_i)\big) = 0$ to obtain

$$b_i = {\rho_1}^{e_{i1}} {\rho_2}^{e_{i2}} \cdots {\rho_n}^{e_{in}}.$$

For each b_i, perform repeated divisions by ρ_j's to obtain the corresponding exponent vector $\vec{e}i = (e_{i1}, e_{i2}, \ldots, e_{in})$.

(3) Determine the coefficients c_i by solving the transposed Vandermonde system

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ b_1 & b_2 & \cdots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ {b_1}^{t-1} & {b_2}^{t-1} & \cdots & {b_t}^{t-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix}. \tag{2.2}$$

The algorithm consists of the following two almost-independent stages:

(1) & (2) ... exact determination of monomials (exponent vectors) existent in P, where the feedback connection polynomial for the value sequence at special evaluation points plays an essential role, and

(3) ... determination of their coefficients in P.

Table 1 summarizes the stepwise complexity of the algorithm. For more detailed descriptions, methods for improvement and their analyses, refer to the original paper and Kaltofen and Lakshman (1988).

In Kaltofen *et al.* (1990), it was reported that step (1) in the above algorithm suffers from intermediate coefficient growth, and the algorithm was extended to a probabilistic modular version in a straightforward manner, as well as for rational polynomial interpolation. The modular algorithm performs all the above steps, except prime factorizations of b_i's in step (2), over $\mathbb{Z}/(p^m)$ for sufficiently large p^m. This causes a chance of failure in solving the Toeplitz system, although according to Kaltofen *et al.* (1990, Lemma 1), its probability can be high enough for practical use.

2.3. MORE MATHEMATICAL FACTS—MAIN IDEA

Let p denote a prime $\in \mathbb{Z}$. For a prime p, we denote the field $\mathbb{Z}/(p)$ by $\mathbb{Z}_p$, and let $\xi \in \mathbb{Z}_p$ be a primitive $(p-1)$st root of unity, i.e., $\xi^k \not\equiv 1 \bmod p$ for $1 \le k < p-1$, which implies $\xi^i \not\equiv \xi^j$ for $i \ne j$. Suppose we evaluate the polynomial

$$a_k \Leftarrow P(\xi^{ku_1}, \xi^{ku_2}, \ldots, \xi^{ku_n}) = \sum_{i=1}^{t} c_i \xi^{k\vec{u}\cdot\vec{e}i} \in \mathbb{Z}_p, \qquad k = 0, 1, \ldots, 2\tau - 1, \tag{2.3}$$

for some $\vec{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{Z}^n$. Notice that the polynomial can be evaluated often more easily and efficiently over $\mathbb{Z}_p$ than over $\mathbb{Z}$. In what follows, we obtain the explicit form of the feedback connection polynomial $\Lambda_p(z)$ of smallest degree for this sequence, and explain its role in determining a set of exponent vectors $\vec{e}i$ for P.

If all values $\vec{u}\cdot\vec{e}i \bmod (p-1)$ are distinct from each other, then

$$\Lambda_p(z) = \prod_{\substack{1 \le i \le t \\ c_i \bmod p \ne 0}} (z - \xi^{\vec{u}\cdot\vec{e}i \bmod (p-1)}), \tag{2.4}$$

by Lemma 2.1. This gives the true number of terms in $(P \bmod p)$, and each discrete logarithm with base ξ in $\mathbb{Z}_p$ of every solution to $\Lambda_p(z) = 0$ gives $(\vec{u}\cdot\vec{e}i \bmod (p-1))$. Under the given degree bounds for P, Kronecker's technique makes all $\vec{u}\cdot\vec{e}i$ distinct by

Table 1. Stepwise time complexity of the Ben-Or and Tiwari algorithm.

	Step	Straightforward	⟶		⟶	Asymptotically fast
(1)	rank(A_τ) & Toeplitz	$O(\tau^3)$	$\xrightarrow{\text{(BM)}}$	$O(\tau^2)$	$\xrightarrow{\text{(KL-T)}}$	$O(M(\tau)\log\tau)$
(2)	$\Lambda(z) = 0$	$O(t^3 dn \log n)$	$\xrightarrow{\text{(AHU)}}$			$O(ndtM(t)\log t \log n)$
(3)	Vandermonde	$O(t^3)$	$\xrightarrow{\text{(Z)}}$	$O(t^2)$	$\xrightarrow{\text{(AHU)}}$	$O(M(t)\log t)$

Here, $M(k)$ denotes the complexity of multiplying two univariate polynomials of degree k at most. All solutions to $\Lambda(z) = 0$ are to be obtained by the p-adic algorithm of Loos (1983). Parenthesized notes above arrows indicate the applications of the following algorithms.

(BM) The Berlekamp/Massey algorithm to directly obtain $\Lambda(z)$ (Blahut, 1985).
(KL-T) The efficient algorithm for rank in Kaltofen and Lakshman (1988), and efficient algorithms to solve Toeplitz systems such as in Brent *et al.* (1980).
(AHU) Asymptotically fast multipoint evaluation algorithm (Aho *et al.*, 1974).
(Z) The efficient algorithm in Zippel (1990) to invert transposed Vandermonde systems.

letting $\vec{u} = \vec{D}$. Thus, if we use such a prime p as $(p-1) > \prod_{i=1}^{n} \bar{d}_i \geq \max_i\{\vec{u} \cdot \vec{e}i\}$, we can exactly determine all exponent vectors existent in $(P \bmod p)$. Another method to make all $\vec{u} \cdot \vec{e}i$ distinct is due to Zippel (1990). According to Lemmas 1 through 3 and Proposition 11 in Zippel (1990), if $p > k_{max}$, where $k_{max} = (n-1)\tau(\tau-1)/2+1$, for at least one of $\vec{u} = (1, k, k^2, \ldots, k^{n-1})$, $1 \leq k \leq k_{max}$, each $\vec{u} \cdot \vec{e}i$ must take a distinct value. Therefore, we can determine the exponent vectors in $(P \bmod p)$ by using the $\Lambda_p(z)$ of the maximum degree among those for the sequences with $\vec{u} = (1, k, k^2, \ldots, k^{n-1})$, $1 \leq k \leq k_{max}$, which requires $2\tau k_{max}$ evaluations of P. This alternative method may be used only when $\prod_i \bar{d}_i$ is too large but k_{max} is sufficiently small for the choice of a single modulus p, on the assumption that P can be evaluated very cheaply. However, such a particular case rarely occurs, and the method is not further treated in this paper.

Next, we consider the cases where $p \leq D_{n+1}$ and the values $b_i = (\xi^{\vec{D} \cdot \vec{e}i} \bmod p)$ are not necessarily distinct. We extend LEMMA 2.1 for the general cases where it may occur that $b_i = b_j$ for some distinct i and j. We define a set σ_i of indices for monomials whose values are all equal to b_i as follows:

$$\sigma_i = \big\{k \mid (1 \leq k \leq t) \wedge (b_i = b_k)\big\},$$

and let $C_i = \sum_{k \in \sigma_i} c_k$. We further define a new set of indices:

$$S = \big\{k \mid (1 \leq k \leq t) \wedge ({}^{\forall}j < k,\, j \notin \sigma_k) \wedge (C_k \neq 0)\big\}.$$

Then, the values a_k act as if P consists only of those monomials whose indices j are $\in S$ with coefficients C_j. Actually, $a_k = \sum_i c_i {b_i}^k = \sum_{j \in S} C_j {b_j}^k$, and for distinct $i, j \in S$, $b_i \neq b_j$. Then, by LEMMA 2.1, the polynomial $\prod_{j \in S}(z - b_j)$ is the unique feedback connection polynomial of smallest degree for the value sequence of a_k's. This polynomial is equal to the square-free part of $\prod_{1 \leq i \leq t,\, C_i \neq 0}(z - b_i)$. This proves the following corollary to LEMMA 2.1.

COROLLARY 2.2. *Let $a_k = \sum_{i=1}^{t} c_i\, {b_i}^k$, and let σ_i and C_i be as above. Then,*

$$\Lambda(z) = \frac{\bar{\Lambda}(z)}{\gcd(\bar{\Lambda}(z), \bar{\Lambda}'(z))} \qquad \text{where } \bar{\Lambda}(z) = \prod_{\substack{1 \leq i \leq t \\ C_i \neq 0}} (z - b_i) \tag{2.5}$$

is a unique feedback connection polynomial of smallest degree for the sequence a_k, $k = 0, 1, \ldots, N$, where $N \geq 2t - 1$.

In our case that $b_i = \xi^{E_i} \in \mathbb{Z}_p$, $\Lambda(z) \in \mathbb{Z}_p[z]$ determines a set of $(E_i \bmod (p-1))$ in P whose corresponding C_i is not equal to 0 modulo p. To determine each value E_i in $\mathbb{Z}$, we need those values for multiple moduli p_j. However, there arises a problem of combinatorial explosion, just as in the case of finding the integer root(s) of a polynomial from the multiple modular images (Loos, 1983). We can hardly determine the exact set of E_i in P from the sets $\{E_i \bmod (p_j - 1)\}$ for multiple p_j. In order to overcome this difficulty, we relax our requirements so that *a set of monomials determined in the first stage would simply give candidates and may include monomials with zero coefficients in* P, as prescribed before. Furthermore, the following well-known theorem (Knuth, 1981) is useful for making appropriate combinations of $E_i \bmod (p_j - 1)$ and reducing the number of monomial candidates.

(Generalized Chinese remainder theorem). *Let $v_1, v_2, \ldots, v_s$ be positive integers. Consider arbitrary integers a and $u_1, u_2, \ldots, u_s$. If u_k's satisfy the following congruences:*

$$u_i \equiv u_j \bmod \gcd(v_i, v_j), \qquad 1 \le i < j \le s, \tag{2.6}$$

then there exists a unique integer u such that $a \le u < a + \mathrm{lcm}(v_1, v_2, \ldots, v_s)$, and $u \equiv u_j \bmod v_j$, $1 \le j \le s$. Otherwise, there exists no such integer.

Suppose $u \Leftarrow E_i$, $a \Leftarrow 0$, $v_j \Leftarrow p_j - 1$ and $u_k \Leftarrow E_i \bmod (p_k - 1)$. This theorem says that, if $\mathrm{lcm}_j\{p_j - 1\} \ge D_{n+1}$, we can recover the value E_i from an appropriate combination of u_k's for different p_j's, and that an arbitrary combination is not allowed to represent any E_i. The condition (2.6) gives a tight constraint on making valid combinations. The constructive proof of the above theorem gives an algorithm to obtain the value $u = E_i$, and each E_i may be converted to the corresponding $\vec{e}i$ easily. Section 3.2 presents a better method to compute $\vec{e}i$ directly from the modular images.

Note that if $\tau C < p_j$ for all j, then none of $(C_i \bmod p_j)$ can be 0, and the above-mentioned relaxed approach is always successful and gives P exactly. Otherwise, the set of candidates may miss those exponent vectors existent in P whose corresponding coefficients C_i are equal to 0 modulo p_j, and our algorithm becomes probabilistic. Because the probability that none of T coefficients is 0 modulo p_j is $(1 - 1/p_j)^T$, the probability of success of our relaxed algorithm is at least $\prod_j (1 - 1/p_j)^t$ and can be sufficiently high if p_j are chosen as $\gg t$.

Note also that the above process for determining a set of monomial candidates is free from the intermediate coefficient growth.

3. Algorithmic Tools

3.1. Factorization of feedback connection polynomials

The explicit expression of the feedback connection polynomial $\Lambda_p(z)$ of smallest degree for the sequence $(a_k = \sum_i c_i \xi^{kE_i} \bmod p)$ has been presented in the previous section. This polynomial can be obtained efficiently by the Berlekamp/Massey algorithm over $\mathbb{Z}_p$ from the sequence with $O(\tau^2)$ operations in $\mathbb{Z}_p$. The role of $\Lambda_p(z)$ is to give all the values $E_i \bmod (p-1)$, which can be done by the following two steps:

(a) complete factorization of $\Lambda_p(z)$ into linear factors: $\Lambda_p(z) = \prod_i (z - b_i)$, and
(b) calculation of the discrete logarithm $(E_i \bmod (p-1))$ of each b_i.

For a large p, this factorization may be efficiently done by the randomized algorithms (Berlekamp, 1970; Cantor and Zassenhaus, 1981), based on

$$z^{p-1} - 1 \equiv (z^{(p-1)/2} - 1)(z^{(p-1)/2} + 1) \bmod p.$$

However, if this congruence is regarded as separating factors by even or odd numbers of the logarithms of their zeros, the above two independent steps seem redundant.

Assume that p is chosen so that $p - 1$ is highly composite, and let G be a divisor of $p - 1$. Then, the above congruence can be generalized to

$$z^{p-1} - 1 \equiv \prod_{k=0}^{G-1} (z^{(p-1)/G} - \xi^{k(p-1)/G}) \bmod p.$$

Each factor in the right-hand side is the product of those linear factors $(z - \alpha)$ whose α satisfies $\log_\xi \alpha \equiv k \bmod G$. Therefore, a non-trivial $\gcd(\Lambda_p(z), z^{(p-1)/G} - \xi^{k(p-1)/G})$ gives the product of only such factors in $\Lambda_p(z)$, and the repeated GCD calculation with k results in a partial factorization of $\Lambda_p(z)$ separated depending on the value of $(E_i \bmod G)$. Figure 1 gives a complete description of a sequential algorithm for this factorization.

Procedure **DistinctOrderFactor**

Input: a prime p, a divisor G of $(p-1)$, a primitive $(p-1)$st root ξ of unity $\in \mathbb{Z}_p$, and a polynomial $\Phi(z)$ over $\mathbb{Z}_p$ which is known to be a product of linear factors.

Output: partial factorization of $\Phi(z)$ given as a set of pairs of a discrete logarithm k and a polynomial factor $\phi_k(z) = \gcd(\Phi(z), z^{(p-1)/G} - \xi^{k(p-1)/G})$.

```
Q ⇐ (p − 1)/G;      A ⇐ ξ^Q mod p;      B ⇐ z^Q mod Φ(z);
if deg Φ(z) = 1, e.g., Φ(z) = z − α then return {[log_A(α^Q), Φ(z)]};
L ⇐ {};      k ⇐ 0;      C ⇐ Φ(z);
while k < G − 1 and C ≠ 1 do
{  if deg B = 0 then return{[log_ξ B, C]} ∪ L;
   w ⇐ gcd(C, B − A^k) mod p;
   if w ≠ 1 then
   {     L ⇐ {[k, w]} ∪ L;   % add a pair [k, w] to L.
         C ⇐ C/w mod p;   % remove the new factor.
         if deg C = 1, e.g., C = z − α then return {[log_A(α^Q), C]} ∪ L;
         if C ≠ 1 then B ⇐ B mod C;   % further simplify z^((p−1)/G)
   }
   k ⇐ k + 1;
}
return L;
```

Figure 1. Procedure DistinctOrderFactor.

To determine each value $E_i \bmod (p-1)$, we have only to apply the above algorithm to partial factors recursively with all relatively prime divisors G_j of $p - 1$ to obtain $\vec{\varepsilon i} = (\varepsilon_{i1}, \varepsilon_{i2}, \ldots, \varepsilon_{is})$, where $\varepsilon_{ij} = E_i \bmod G_j$.

3.2. RECOVERY OF EXPONENT VECTORS

Once we have obtained $\vec{\varepsilon}_i$, we convert $\vec{\varepsilon}_i$ to its mixed-radix representation $\vec{e i}$ with $\vec{d} = (d_1, d_2, \ldots, d_{n-1})$ of E_i. This conversion can be done without computing E_i and only with integers of a fixed magnitude by the following two steps:

(1) convert $\vec{\varepsilon}_i$ to a mixed-radix representation $\vec{\delta i}$ with $\vec{q} = q_1, q_2, \ldots, q_{s-1}$, where

$$q_j = \begin{cases} G_1 & \text{for } j = 1, \\ \operatorname{lcm}(G_1, G_2, \ldots, G_j) / \operatorname{lcm}(G_1, G_2, \ldots, G_{j-1}) & \text{for } j \geq 2, \end{cases}$$

and then,

(2) convert $\vec{\delta i}$ to an exponent vector $\vec{e i}$ by performing radix conversion from $\vec{q}$ to $\vec{d}$.

Algorithms for these steps are well known (Knuth, 1981). The former conversion requires $r_j = G_j/q_j$ and A_{kj} such that $g_{kj} = \gcd(G_k, G_j) \equiv A_{kj} G_k \bmod G_j$, which are common for all $\vec{\varepsilon}_i$ and can be obtained with $O(s^2 \log p)$ operations in $\mathbb{Z}_p$. With this precomputation, each $\vec{\varepsilon}_i$ can be converted to $\vec{\delta i}$ and then to $\vec{e i}$ with respective costs $O(s^2)$ and $O(sn)$.

3.3. SOLVING TRANSPOSED VANDERMONDE SYSTEMS

The linear system (2.2) can be solved efficiently by Zippel's algorithm (Zippel, 1990) using auxiliary polynomials $B(z) = \prod_{i=1}^{t}(z - b_i)$ and $D(z) = \sum_{k=0}^{t-1} a_k z^{t-k}$. In our case, the algorithm is applied over $\mathbb{Z}_p$, and notice $B(z)$ is given by $\Lambda_p(z)$.

4. New Modular Algorithm

We now give a complete description of our new modular algorithm for sparse multivariate polynomial interpolation. The behavior of the algorithm differs depending on the magnitude of the value D_{n+1} as explained in Section 2.3.

4.1. DETERMINISTIC ALGORITHM: FOR MODERATE-SIZED D_{n+1}

Assume the bound D_{n+1} is sufficiently small that there exists a prime p (of machine word size) such that $p > D_{n+1}$. In this case, the value $E_i \bmod (p-1)$ is equal

Algorithm **SmallCase1**

Input : a prime $p = G_1 G_2 \cdots G_s + 1$, where G_j's are not necessarily prime but are pairwise relatively prime, an n-variate black-box polynomial $P(x_1, x_2, \ldots, x_n)$ which allows evaluations at arbitrary points, a bound τ for the number of terms in P, and the product D_i's of the degree bound d_j's.

Output : The polynomial representation of $P \bmod p = \sum_{k=1}^{\bar{t}} (c_k \bmod p)\, m_k$.

(**S1-1**) Let ξ be a primitive $(p-1)$st root of unity in $\mathbb{Z}_p$.

(**S1-2**) Evaluate and let $a_k \Leftarrow P(\xi^{kD_1}, \xi^{kD_2}, \ldots, \xi^{kD_n}) \bmod p$, for $k = 0, 1, \ldots, 2\tau - 1$.

(**S1-3**) Apply the Berlekamp/Massey algorithm to this value sequence a_k to obtain $\Lambda_p(z)$. Let $\bar{t} \Leftarrow \deg(\Lambda_p(z))$, which is equal to the number of terms in $(P \bmod p)$.

(**S1-4**) $L \Leftarrow \{\, [\ [\ ], \Lambda_p(z)\ ]\,\}$;
for j **:=** s **step** (-1) **to** 1 **do** the following:
(S1-4.1) $L_{new} \Leftarrow \{\}$;
(S1-4.2) For each pair [ε-*list*, $\phi(z)$] in L, apply **DistinctOrderFactor** to $\Phi(z) = \phi(z)$ with $G = G_j$ to obtain a set w of pairs [$k, \phi_k(z)$]'s.
(S1-4.3) For each pair [$k, \phi_k(z)$] in w, $L_{new} \Leftarrow \{$ [[k, ε-*list*], $\phi_k(z)$] $\} \cup L_{new}$;
(S1-4.4) $L \Leftarrow L_{new}$;

(**S1-5**) For all i and j such that $1 \le i < j \le s$, apply Euclid's algorithm to G_i and G_j to obtain A_{ij} such that $1 \equiv A_{ij} G_i \bmod G_j$.

(**S1-6**) For each pair [$\vec{\varepsilon}$, $\phi(z)$] in L, convert $\vec{\varepsilon}$ to $\vec{e}$ using A_{ij}, $g_{ij} = 1$, $q_j = G_j$ and $r_j = 1$.
% Each $\vec{e}k = \vec{e}$ defines a monomial $m_k = x_1^{e_{k1}} x_2^{e_{k2}} \cdots x_n^{e_{kn}}$, $k = 1, 2, \ldots, \bar{t}$.

(**S1-7**) Letting $b_i = \xi^{E_i}$ for every monomial m_i, solve a transposed Vandermonde system to obtain the coefficients $(c_i \bmod p)$'s.

Algorithm **SmallCase**

Input : an n-variate black-box polynomial $P(x_1, x_2, \ldots, x_n)$ over $\mathbb{Z}$, a bound τ for the number of terms in P, a coefficient bound C, and the product D_i's, as above.

Output : The polynomial representation of $P = \sum_{k=1}^{t} c_k\, m_k$.

(**S-1**) Choose a sufficient number of primes $p_1, p_2, \ldots, p_N$ such that $p_i \ge D_{n+1}$ and $\prod_{i=1}^{N} p_i \ge 2C$.

(**S-2**) For each p_i, $i = 1, 2, \ldots, N$, use the above **SmallCase1** with $p = p_i$ to obtain the modular image polynomial $P \bmod p_i$.

(**S-3**) Apply the Chinese remainder algorithm to $(P \bmod p_i)$'s to obtain P.

Figure 2. Deterministic algorithm for polynomial interpolation when D_{n+1} is moderate-sized.

to E_i, because $E_i = e_{i1}D_1 + e_{i2}D_2 + \cdots + e_{in}D_n < D_{n+1} < p$. As explained in Section 2.3, the feedback connection polynomial $\Lambda_p(z)$ of (2.4) obtained by applying the Berlekamp/Massey algorithm to the sequence (2.3) determines all those values whose corresponding exponent vectors $\vec{e}i$ are existent in $(P \bmod p)$.

The subalgorithm **SmallCase1** of Figure 2 obtains the polynomial $(P \bmod p)$. In that, assuming that $p-1$ is highly composite, e.g., $p = G_1G_2\cdots G_s + 1$, where G_j's are not necessarily prime but are pairwise relatively prime, step (S1-4) separates the factors of Λ_p by applying **DistinctOrderFactor** to each partial factor ϕ of Λ_p repeatedly for each $G = G_j$, to finally obtain every linear factor with its corresponding $\vec{\varepsilon}$. Step (S1-6) converts each $\vec{\varepsilon}$ to the true exponent vector $\vec{e}$ by the algorithm described in Section 3.2, without computing the value $E = \vec{D} \cdot \vec{e}$.

If $2C < p$, the polynomial obtained by **SmallCase1** gives P itself. However, in general, the polynomial P over $\mathbb{Z}$ must be recovered from its modular images by sufficiently many moduli. The algorithm **SmallCase**, taking the magnitude of the coefficients into account and using the Chinese remainder theorem, obtains the polynomial P over $\mathbb{Z}$ deterministically. Note that the utilization in the repeated invocations of the subalgorithm of the exponent vectors fixed by the previous invocations will improve the efficiency.

4.2. RELAXED ALGORITHM: FOR LARGE D_{n+1}

In this case, we must use multiple primes p_k as moduli. For each p_k, the feedback connection polynomial (2.5), or more precisely all solutions b_i to the polynomial, will determine a set of $(E_i \bmod (p_k - 1))$ in P whose corresponding C_i is not equal to 0 modulo p_k. If none of C_i is 0 modulo p_k, these values definitely reflect a complete set of values E_i over $\mathbb{Z}$ in P, but what combination of the values corresponds to the true E_i? Let S_k denote a set of those values for p_k:

$$S_k = \{E_i \bmod (p_k - 1) \mid 1 \le i \le t\}.$$

Then, according to the generalized Chinese remainder theorem, for $u_j \in S_j$ and $u_k \in S_k$ to correspond to a single E_i, $(u_j - u_k)$ must be divisible by $\gcd(p_j - 1, p_k - 1)$. We shall use this condition to make a set of candidates for E_i, only for circularly neighboring p_k's.

Let p_k, $k = 1, 2, \ldots, s$ be primes (of machine word size) chosen so that

(C1) $\prod_{k=1}^{s} p_k > 2C$,
(C2) $\operatorname{lcm}(G_1, G_2, \ldots, G_s) > D_{n+1}$, where G_j is defined by

$$G_j = \begin{cases} \gcd(p_j - 1, p_{j+1} - 1), & 1 \le j < s, \\ \gcd(p_s - 1, p_1 - 1), & j = s, \end{cases}$$

and $G_0 = G_s$, and
(C3) $p_k \gg \tau \ge t$.

For each p_k, we compute the feedback connection polynomial and factorize it with $G = G_{k-1}$ and G_k using **DistinctOrderFactor**, to obtain the pairs

$$\pi_k^{(i)} = [E_i \bmod G_{k-1}, E_i \bmod G_k] = [\varepsilon_{ik-1}, \varepsilon_{ik}].$$

The algorithm **LargeCase1** of Figure 3 returns these pairs along with the corresponding partial factors $\phi_{k_{12}}$ of a feedback connection polynomial. Notice if $\operatorname{lcm}(G_1, G_2) < p - 1$, the factor $\phi_{k_{12}}(z)$ may further separate, i.e., $\deg \phi_{k_{12}}(z) > 1$, which indicates that there

Algorithm **LargeCase1**

Input : a prime p, a primitive $(p-1)$st root ξ of unity in $\mathbb{Z}_p$, an n-variate black-box polynomial $P(x_1, x_2, \ldots, x_n)$ which allows evaluations at arbitrary points, a bound τ for the number of terms in P, the product D_i's of the degree bound d_j's, and the divisors G_1 and G_2 of $p-1$.

Output : $\{a_k\}$, $\bar{t}$ and such a set L as described below.

(**L1-1**) *the same as* **(S1-1)** *and* **(S1-2)** *of* **SmallCase1**.

(**L1-2**) Apply **DistinctOrderFactor** to $\Phi = \Lambda_p$ with $G = G_1$ to obtain a set S_1 of $[\ k_1, \phi_{k_1}(z)\ ]$'s.

(**L1-3**) For each pair $[\ k_1, \phi_{k_1}(z)\ ]$ in S_1, apply **DistinctOrderFactor** to $\phi_{k_1}(z)$ with $G = G_2$ to obtain a set of $[\ k_2, \phi_{k_{12}}(z)\ ]$'s, and collect every $[[\ k_1, k_2\ ],\ \phi_{k_{12}}(z)\ ]$ in L.

(**L1-4**) **return** L;

Algorithm **LargeCase**

Input : an n-variate black-box polynomial $P(x_1, x_2, \ldots, x_n)$ over $\mathbb{Z}$, a bound τ for the number terms in P, a coefficient bound C, degree bound d_i's and the product D_i's.

Output : The polynomial representation of $P = \sum_{k=1}^{t} c_k\, m_k$.

(**L-1**) Choose a sufficient number of primes $p_1, p_2, ..., p_s$ so that they satisfy conditions **(C1)**, **(C2)** and **(C3)**. Let ξ_k be a primitive $(p_k - 1)$st root of unity in $\mathbb{Z}_{p_k}$.

(**L-2**) For each prime p_k, $k = 1, 2, \ldots, s$, use the above **LargeCase1** with $p = p_k$, $\xi = \xi_k$, $G_1 = G_{k-1}$ and $G_2 = G_k$ to obtain $\bar{t_k} \Leftarrow \bar{t}$ and a set $L_k = \{\ [[\ \varepsilon_{k-1}, \varepsilon_k\ ],\ \phi_k(z)\ ]\ \}$. Note that the value sequence a_i's computed in **LargeCase1** is to be used in step (L-6.1) to determine the coefficients.

(**L-3**) From the sets L_k, make every possible combination of $\vec{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_s)$, and form a set L of the combinations, $L = \{\ [\ \vec{\varepsilon},\ [\ \phi_1(z), \phi_2(z), \ldots, \phi_s(z)\ ]]\ \}$.

(**L-4**) Prepare g_{ij}'s, A_{ij}'s, q_i's and r_i's.

(**L-5**) For each pair $[\ \vec{\varepsilon},\ [\ \phi_1(z), \phi_2(z), \ldots, \phi_s(z)\ ]]$ in L,

- (L-5.1) Convert $\vec{\varepsilon}$ to a mixed radix representation $\vec{e} = (e_1, e_2, \ldots, e_n)$ with $(d_1, d_2, \ldots, d_{n-1})$,
- (L-5.2) dispose the pair if $\vec{\varepsilon}$ does not satisfy the degree bound condition with $\bar{d_j}$'s,
- (L-5.3) dispose the pair if $\phi_k(b_k) \not\equiv 0 \bmod p_k$ for some k, where $b_k = {\xi_k}^{\vec{D}\cdot\vec{e}} \bmod p_k$,
- (L-5.4) and collect $[\ \vec{e},\ [\ b_1, b_2, \ldots, b_s\ ]]$ in S.

(**L-6**) For each prime p_k, $k = 1, 2, \ldots, s$,

- (L-6.1) Solve the transposed Vandermonde system with non-duplicated b_i's to obtain the coefficients $(C_i \bmod p_k)$.
- (L-6.2) If no duplication is found $(|\ L\ | = \bar{t_k})$, $P \bmod p_k$ is obtained,
- (L-6.3) otherwise, for each $\vec{e}$ with duplicated b_i's, create a non-singular linear system for their coefficients by newly evaluating P, and solve it to determine the coefficients $\bmod p_k$.

(**L-7**) Recover the polynomial $P(x_1, x_2, \ldots, x_n)$ from its modular images, as in **(S-3)** of **SmallCase**.

Figure 3. Algorithm for polynomial interpolation when D_{n+1} is large.

exist E_i and E_j such that $E_i \equiv E_j \bmod G_1$ and $\bmod G_2$, but $E_i \not\equiv E_j \bmod (p-1)$. Then, the valid combinations of $\pi_k^{(i)}$'s are only those combinations in which the second element of $\pi_k^{(i)}$ is equal to the first element of $\pi_{k+1}^{(j)}$. In this way, we form all possible combinations $\vec{\varepsilon_i}$ from $\pi_k^{(i)}$'s (step (L-3)), and use them as a set of candidates of $\vec{\varepsilon_i}$ for P. The condition **(C2)** guarantees that every possible value of E_i takes a different $\vec{\varepsilon_i}$, and each $\vec{\varepsilon_i}$ can be converted to the true exponent vector $\vec{e}i$ by the algorithm described in Section 3.2 (L-5.1). Note that the set of candidates may contain $\vec{\varepsilon_i}$ not existent in P.

For a single modulus p, it may occur that $b_i \equiv b_j \bmod p$, i.e., $E_i \equiv E_j \bmod (p-1)$ for $i \neq j$. For every such E_i, the solution to the transposed Vandermonde system can give only the sum $(C_i \bmod p_k)$ (L-6.1) of those coefficients c_j's whose corresponding E_j is congruent to E_i modulo p_k. To fix such terms, we need only solve another linear system

for a limited number of monomial candidates obtained by further evaluations of P at appropriate points (L-6.3).

Another problem is the number of candidates $\vec{\varepsilon}_i$ made from $\pi_k^{(i)}$'s. The candidate set may contain those $\vec{\varepsilon}_i$ whose corresponding exponent vectors do not exist in P. Such redundant exponent vectors may be disposed of by checking against the degree bound conditions with $\bar{d}_j$ (L-5.2), or by checking if their corresponding b_i's satisfy $\Lambda_{p_k}(b_i) \equiv 0$ (L-5.3). Furthermore, non-neighboring $\gcd(p_i - 1, p_j - 1) \neq 1$ may be used for pruning. Anyway, it is expected that under the condition **(C3)**, E_i's will distribute very sparsely and the number of such exponent vectors will be quite limited.

As noted in Section 2.3, the algorithm is probabilistic unless $\tau C < p_k$ for all k. The condition **(C3)** is set also to attain a higher probability of success.

5. Examples

To help understand our algorithm, we shall give a simple example. Suppose we determine (the exponent vectors in) the following polynomial:

$$P(x_1, x_2, x_3) = {x_1}^{30}{x_2}^{5}{x_3}^{7} + 3{x_1}^{5}{x_3}^{10} - {x_2}^{7}{x_3}^{3} + {x_1}^{6}{x_2}^{20},$$

thus $\tau \geq t = 4$, and we use $(d_1, d_2, d_3) = (31, 23, 11)$.

USING THE DETERMINISTIC ALGORITHM

Let $p = 227951 \geq \prod_i d_i = 31 \times 23 \times 11$ and let $\xi = 11$. The factorization of the feedback connection polynomial for the sequence $P(11^j, 11^{31j}, 11^{31\times 23\times j})$, $j = 0, 1, \ldots, 7$, gives

$$\Lambda_p(z) = (z - \xi^{5176})(z - \xi^{7135})(z - \xi^{2356})(z - \xi^{626}).$$

Because $5176 = 30{+}31{\times}(5{+}23{\times}7)$, $7135 = 5{+}31{\times}(0{+}23{\times}10)$, $2356 = 0{+}31{\times}(7{+}23{\times}3)$ and $626 = 6 + 31 \times (20 + 23 \times 0)$, we can determine all the above exponent vectors. Use of factors of $p - 1 = 2 \times 5^2 \times 47 \times 97$ may speed up the above process, especially the factorization of $\Lambda_p(z)$.

USING THE RELAXED ALGORITHM

We shall use three primes $p_1 = 1327$ $(= 2 \times 3 \times 13 \times 17 + 1)$, $p_2 = 4447$ $(= 2 \times 3^2 \times 13 \times 19 + 1)$ and $p_3 = 3877$ $(= 2^2 \times 3 \times 17 \times 19 + 1)$. The feedback connection polynomials $\Lambda_p(z)$ for the sequence $P(\xi^j, \xi^{31j}, \xi^{31\times 23j})$, $j = 0, 1, \ldots, 7$, are as follows.

p_k	ξ	$\Lambda_p(z)$
1327	3	$(z - 3^{1198})(z - 3^{505})(z - 3^{1030})(z - 3^{626})$
4447	3	$(z - 3^{730})(z - 3^{2689})(z - 3^{2356})(z - 3^{626})$
3877	2	$(z - 2^{1300})(z - 2^{3259})(z - 2^{2356})(z - 2^{626})$

With $G_1 = 2 \times 3 \times 13$, $G_2 = 2 \times 3 \times 19$ and $G_3 = 2 \times 3 \times 17$, the factorizations of Λ_p, by two times applications of **DistinctOrderFactor** for each, will give the following $\pi_k^{(i)}$.

k	p_k	$\pi_k^{(i)}(E_i \bmod (p_k - 1))$
1	1327	[97;37](505), [14; 2](626), [10;16](1030), [76;28](1198)
2	4447	[2;56](626), [28;46](730), [16;76](2356), [37;67](2689)
3	3877	[56;14](626), [46;76](1300), [76;10](2356), [67;97](3259)

Only the following combinations of $\pi_k^{(i)} = [\varepsilon_{ik-1}, \varepsilon_{ik}]$ are valid:

	$\vec{\varepsilon}_1$	$\vec{\varepsilon}_2$	$\vec{\varepsilon}_3$	$\vec{\varepsilon}_4$
modG_1	37	2	16	28
modG_2	67	56	76	46
modG_3	97	14	10	76

We convert each $\vec{\varepsilon}_i$ to a mixed-radix representation $\vec{\delta}$ with $(q_1, q_2, q_3) = (2\times3\times13, 19, 17)$, and then to the one $\vec{e}i$ with $(d_1, d_2, d_3) = (31, 23, 11)$.

$\vec{\varepsilon}_i$: $(37, 67, 97)$, $(2, 56, 14)$, $(16, 76, 10)$, $(28, 46, 76)$ ... remainders
↓ conversion to mixed-radix representations with (q_1, q_2, q_3)
$\vec{\delta}_i$: $(37, 15, 4)$, $(2, 8, 0)$, $(16, 11, 1)$, $(28, 9, 3)$
↓ conversion of mixed-radix, $(q_1, q_2, q_3) \to (d_1, d_2, d_3)$
$\vec{e}i$: $(5, 0, 10)$, $(6, 20, 0)$, $(0, 7, 3)$, $(30, 5, 7)$ ... exponent vectors

6. Efficiency

6.1. COMPLEXITY

In this section, we shall give a very brief analysis of the stepwise time complexity of our algorithms, and later consider how parallelization is to be applied. We limit our concern to practical efficiency, and we do not consider the use of asymptotically fast algorithms. In the following, the measure of the time complexity is the number of arithmetic operations in $\mathbb{Z}_p$, unless otherwise noted.

Consider the complexity of **DistinctOrderFactor** (hereinafter, abbreviated as **DOF**). Let t be the degree of $\Phi(z)$. The calculation of every discrete logarithm can be done in $O(G)$ operations, and may be performed t times at most. The calculations of (z^Q mod $\Phi(z)$) and of the gcd's in the loop dominate, and their respective complexities are bounded by $O(t^2 \log p)$ and $O(Gt^2)$.

Next, consider the **SmallCase** algorithms. The algorithm **SmallCase1** reveals clear correspondence with the original Ben-Or and Tiwari algorithm, and steps (S1-3), (S1-6) and (S1-7) can be accomplished in $O(\tau^2)$, $O(t(s^2+ns))$ and $O(t^2)$ respectively. Step (S1-5) requires $O(s^2 \log p)$ operations, and is thus negligible. The major difference from the original algorithm is the loop of step (S1-4). This step corresponds to the factorization of the feedback connection polynomial over $\mathbb{Z}$ and the prime factorizations of b_i, and the complexities in both algorithms should be almost comparable. Analysis of step (S1-4) is complicated because the degrees of polynomials vary. The first call to **DOF** takes $O(ct^2 + Gt^2)$. Since (the number of polynomials) $\times$ (the average degree of polynomials) can be assumed constant ($\approx t$) within the loop of (S1-4), and since the complexity of **DOF** is quadratic in the degree, reduction of polynomial degrees is beneficial. If the distribution of monomials in P is assumed to be uniformly random, the first call to **DOF** with $G \simeq t$ will give almost complete factorization. This is nearly optimal because subsequent calls to **DOF** with polynomials of very low degree are almost negligible ($O(t)$), and can thus be accomplished in $O(Gt^2) \simeq O(t^3)$. Therefore, the dominant step in **SmallCase1** is the loop of (S1-4) for factorizations.

The analysis of the **LargeCase** algorithms is more complicated. With the factor s

of the number of primes used being ignored, how much do the two calls to **DOF** cost, compared with other $O(t^2)$ steps? If G_1 is chosen $\geq t$, it requires $> O(t^3)$, and the second calls can be ignored. Otherwise, we would have, on average, G_1 polynomials of degree (t/G_1) after the first call. The second calls thus require $O((G_2/G_1)t^2)$, and in total $O((G_1 + G_2/G_1)t^2)$. In practice, $\mathrm{lcm}(G_1, G_2) \simeq p$, which is assumed $\gg t$, and the factor $(G_1 + G_2/G_1)$ would be $> t^{1/2}$. Again, step for factorization is dominant. Notice that under **(C2)**, the degree of each $\phi_k(z)$ is expected to be very small, which means the cost of step (L-5.3) is negligible. Also note that step (L-3) costs $O(t^s)$, which seems dominant, but in practice, its coefficient is very small because it only requires equality checks and can be done very cheaply. The timing data in the next section supports the above analysis.

6.2. EMPIRICAL STUDY: BEHAVIOR IN SEQUENTIAL EXECUTION

The main part of our algorithm was implemented in Rlisp to assess the performance of the steps of our major contribution (from the application of the Berlekamp/Massey algorithm to the recovery of $\vec{e}i$). It was executed on CSL running on a SparcStation-2 (SunOS 4.1.3). All timings are given in `msec`.

The algorithm **SmallCase1** was tested against randomly generated polynomials in $n = 5$ variables, with $p = 227951 = 2 \times 25 \times 47 \times 97 + 1$ ($s = 4$ different G_i's are used), degree bounds $d_i = 11$ ($D_6 = 11^5$), and the number of terms $t = 10, 20, 50, 100, 200$. Table 2 summarizes the timings of our experiments. Each column contains the time for:

B/M ... application of the Berlekamp/Massey algorithm,
DOF ... **DistinctOrderFactor**,
$\vec{\varepsilon} \to \vec{e}$ ... determination of exponent vectors.

Tests of the **LargeCase** algorithm were done with polynomials in $n = 4$ variables, with $d_i = 101$ ($D_5 = 101^4$) and $t = 10, 20, 50, 100, 200$. The following primes

$$p_1 = 227951 = 2 \times 5^2 \times 47 \times 97 + 1,$$
$$p_2 = 489851 = 2 \times 5^2 \times 97 \times 101 + 1,$$
$$p_3 = 998689 = 2^5 \times 3 \times 101 \times 103 + 1,$$
$$p_4 = 348553 = 2^3 \times 3^2 \times 47 \times 103 + 1$$

are used as moduli ($G_1 = 2 \times 5^2 \times 97$, $G_2 = 2 \times 101$, $G_3 = 2^3 \times 3 \times 103$, $G_4 = 2 \times 47$, and the lcm is $2^3 \times 3 \times 5^2 \times 47 \times 101 \times 103 \gg t$). Table 3 summarizes the timings.

Table 2. Stepwise timings (`msec`) of **SmallCase1**.

t	B/M	**DOF**	$\vec{\varepsilon} \to \vec{e}$
10	17	250	16
20	33	900	50
50	200	4117	50
100	700	17300	84
200	2800	57450	166

Table 3. Stepwise timings (`msec`) of **LargeCase**.

t	prime	B/M	**DOF**	$\vec{\varepsilon} \to \vec{e}$
10	p_1	17	2550	17
	p_2	33	49866	
	p_3	17	1551	
	p_4	33	16489	
20	p_1	34	5367	33
	p_2	50	37417	
	p_3	66	3283	
	p_4	34	17600	
50	p_1	217	59867	83
	p_2	216	30966	
	p_3	217	30684	
	p_4	217	25417	
100	p_1	733	257482	150
	p_2	800	169184	
	p_3	800	78883	
	p_4	784	95783	
200	p_1	2750	893583	417
	p_2	2949	777600	
	p_3	3134	320467	
	p_4	2850	366600	

In every case, our algorithm was able to determine the set of exponent vectors exactly. In our implementation, k in **DistinctOrderFactor** was randomly chosen, and we have observed that the timing of **DOF** heavily depends on the sequence of k. In any case, most of the computing time is spent in **DOF**.

6.3. CONSIDERATIONS FOR PARALLELIZATION

There are obviously many places in our algorithm to be processed concurrently, e.g., independent calculations with multiple primes including the final step of the binary applications of the Chinese remainder algorithm, conversions of independent exponent vectors and so on. Among them, factorization is the most significant factor.

In the description of **DOF**, the reductions of B and C in the while-loop appears to disturb the concurrency. However, the iteration steps for finding and separating factors with different k's are basically independent and can be executed in parallel, and the reductions are used only for computational simplification (*algebraic pruning*). Therefore, to parallelize the loop for k, this algebraic pruning process can be omitted. Figure 4 describes such an extension of **DistinctOrderFactor**. In practice, we must consider the trade-off between the benefits from parallelization and from algebraic pruning. When the loop length is much larger than the number of processors and the loop is divided into multiple while-loops to be executed in parallel, algebraic pruning ought to be performed within each loop. Furthermore, if the cost for communication is sufficiently cheap compared with the costs for loop executions, reduction information should be transmitted and algebraic pruning can be effectively used across the parallel loops, asynchronously to each loop. If the outer loop(s) with G_i is(are) taken into account, any incomplete L can be forwarded to the further factorization process in step (S1-4.2) of **SmallCase1** and step (L1-3) of **LargeCase1**. In other words, synchronization for each G_i is not required, and the later gcd calculations with an element from the incomplete L can be pipelined.

$Q \Leftarrow (p-1)/G$; $A \Leftarrow \xi^Q \bmod p$; $B_i \Leftarrow z^Q \bmod \Phi(z)$;
if $\deg \Phi(z) = 1$, e.g., $\Phi(z) = z - \alpha$ **then return** $\{[\log_A(\alpha^Q), \Phi(z)]\}$;
$S \Leftarrow \lceil (G-2)/N \rceil$;
for i := 0 **to** $N-1$ **do_parallel**
  $k_i \Leftarrow S \cdot i$; $G_i \Leftarrow \min(G-1, k_i + S)$; $L_i \Leftarrow \{\}$; $C_i \Leftarrow \Phi(z)$;
  while $k_i < G_i - 1$ and $C_i \neq 1$ **do**
    if $\deg B_i = 0$ **then** { $L_i \Leftarrow \{[\log_\xi B_i, C_i]\} \cup L_i$; **exit_do_parallel**;}
    $w_i \Leftarrow \gcd(C_i, B_i - A^{k_i}) \bmod p$;
    if $w_i \neq 1$ **then** {
      $L_i \Leftarrow \{[k_i, w_i]\} \cup L_i$; % add a pair $[k_i, w_i]$ to L_i.
      $C_i \Leftarrow C_i / w_i \bmod p$; % ← algebraic pruning.
      if $\deg C_i = 1$, e.g., $C_i = z - \alpha$ **then** {
        $L_i \Leftarrow \{[\log_A(\alpha^Q), C_i]\} \cup L_i$; **exit_do_parallel**;}
      if $C_i \neq 1$ **then** $B_i \Leftarrow B_i \bmod C_i$; % ← algebraic pruning.
    }
    $k_i \Leftarrow k_i + 1$;
return $\bigcup_{i=0}^{N-1} L_i$;

Figure 4. Distinct Order Factorization (parallel version).

6.4. EMPIRICAL RESULTS OF PARALLEL FACTORIZATION

Parallel implementation of our algorithm is in progress, with major emphasis placed on speeding up **DOF** by parallelization. Coding is being done in a concurrent logic programming system KLIC (Chikayama *et al.*, 1994), with several subroutines written in C for polynomial manipulations. KLIC is a portable implementation of a concurrent logic programming language KL1, and there are currently three variants of parallel implementation of KLIC: (1) shared-memory, (2) message-passing using socket-based library, and (3) message-passing through shared-memory. The variant (3) is used here to simulate those hardwares which have fast communication paths among processors, and was run on a SPARCcenter 2000 (20 × SuperSPARC @ 40 MHz). Following the consideration in the previous section, three different types of parallelized **DOF** are implemented, and are currently being studied empirically:

(a) with synchronization at the end of the loop,
(b) pipelining without synchronization, and
(c) with synchronization and algebraic pruning.

Table 4 lists the wall-clock times (in `sec`) taken by the problem in Table 3 with $t = 100$. The numbers in parentheses indicate parallel speed-up ratio, and "+1" in the #PE column means a single processor for load-distribution control. We observe that method (b) is slightly better than (a), and method (c) shows the best parallel speed-up increase.

The basic functionality of **DOF** is the search for linear factors, and a parallel search should be effective and improve the average computing time. In order to observe the effect of search parallelization, we have tested the extreme cases with the best and worst

Table 4. Timings (`sec`) by parallel **DOF** in KLIC.

Prime	#PE	(a)	(b)	(c)
p_1	$1+1$	95.4 (1.0)	95.1 (1.0)	95.3 (1.0)
	$2+1$	71.5 (1.3)	71.2 (1.3)	48.6 (2.0)
	$4+1$	36.2 (2.6)	35.9 (2.7)	23.6 (4.0)
	$8+1$	18.3 (5.2)	18.4 (5.3)	12.3 (7.8)
	$12+1$	12.2 (7.8)	12.5 (7.6)	8.6 (11.1)
	$16+1$	9.6 (10.0)	9.3 (10.3)	6.2 (15.3)
p_2	$1+1$	17.3 (1.0)	17.0 (1.0)	17.2 (1.0)
	$2+1$	12.2 (1.4)	12.3 (1.4)	8.7 (2.0)
	$4+1$	8.3 (2.1)	8.4 (2.0)	5.0 (3.4)
	$8+1$	5.4 (3.2)	5.7 (3.0)	3.5 (5.0)
	$12+1$	4.1 (4.3)	4.1 (4.2)	3.1 (5.6)
	$16+1$	3.5 (4.9)	3.5 (4.8)	2.8 (6.2)
p_3	$1+1$	14.6 (1.0)	14.3 (1.0)	14.3 (1.0)
	$2+1$	11.1 (1.3)	10.9 (1.3)	7.3 (2.0)
	$4+1$	8.0 (1.8)	8.7 (1.7)	4.5 (3.2)
	$8+1$	5.0 (2.9)	5.0 (2.8)	3.0 (4.7)
	$12+1$	4.6 (3.1)	3.9 (3.6)	2.7 (5.3)
	$16+1$	3.8 (3.8)	3.5 (4.1)	2.5 (5.8)
p_4	$1+1$	13.0 (1.0)	12.8 (1.0)	12.7 (1.0)
	$2+1$	9.0 (1.5)	8.9 (1.4)	7.6 (1.7)
	$4+1$	6.0 (2.2)	5.8 (2.2)	4.4 (2.9)
	$8+1$	3.8 (3.4)	3.7 (3.4)	3.1 (4.1)
	$12+1$	3.2 (4.0)	3.2 (4.0)	2.8 (4.6)
	$16+1$	3.1 (4.3)	2.7 (4.8)	2.4 (5.3)

search order. Here, the best and the worst mean that all factors are found in the first and the last t iterations in sequential execution. Test cases are as follows:

(d) best case with algebraic pruning,
(e) worst case with algebraic pruning,
(f) best case without algebraic pruning (only for parallel), and
(g) worst case without algebraic pruning (only for parallel).

Table 5 gives the timings of these extreme cases.

We observe, from the comparison of cases (d) and (f), that algebraic pruning is very effective even in a parallel search. This is because, as was noted in Section 6.1, the total computational cost is proportional to the loop length and quadratic in the degrees of the target polynomials. The reduction of the degrees is much more effective than the parallel speed-up in this search problem. The timings of (e) and (g) for the worst cases show (almost) linear speed-up, and clearly indicate the effect of the parallel search.

7. Concluding Remarks

If 32-bit integers are used for moduli, most of practical cases seem to fall into the small case of Section 4.1 after appropriate variable transformations. For such cases, our algorithm is deterministic, and would be efficient because it is free from the problem

Table 5. Timings (sec) of extreme cases by parallel **DOF** in KLIC.

Prime	#PE	(d)	(e)	(f)	(g)
p_1	$1+1$	2.4 (1.0)	143.1 (1.0)	2.3 (1.0)	143.8 (1.0)
	$2+1$	2.3 (1.0)	72.8 (2.0)	71.8 (0.0)	71.7 (2.0)
	$4+1$	2.5 (1.0)	36.4 (4.0)	36.2 (0.0)	36.1 (4.0)
	$8+1$	2.6 (0.9)	18.3 (7.8)	18.1 (0.1)	18.4 (7.8)
	$12+1$	2.4 (1.0)	12.2 (11.7)	12.3 (0.2)	12.1 (11.9)
	$16+1$	2.4 (1.0)	9.4 (15.3)	9.1 (0.3)	9.1 (15.8)
p_2	$1+1$	9.8 (1.0)	25.9 (1.0)	6.4 (1.0)	35.3 (1.0)
	$2+1$	5.7 (1.7)	15.2 (1.7)	10.0 (0.6)	17.1 (2.1)
	$4+1$	3.7 (2.7)	7.8 (3.3)	6.9 (0.9)	10.8 (3.3)
	$8+1$	2.2 (4.4)	4.2 (6.2)	4.3 (1.5)	6.8 (5.2)
	$12+1$	2.9 (3.4)	3.0 (8.6)	4.0 (1.6)	5.8 (6.0)
	$16+1$	2.3 (4.3)	3.8 (6.8)	3.6 (1.8)	5.3 (6.7)
p_3	$1+1$	10.2 (1.0)	27.1 (1.0)	5.8 (1.0)	31.7 (1.0)
	$2+1$	5.6 (1.8)	13.7 (2.0)	10.3 (0.6)	15.0 (2.1)
	$4+1$	4.3 (2.4)	7.1 (3.8)	6.9 (0.9)	10.1 (3.1)
	$8+1$	3.4 (3.0)	4.2 (6.4)	4.3 (1.4)	6.1 (5.2)
	$12+1$	2.8 (3.6)	3.3 (8.3)	4.2 (1.4)	5.3 (6.0)
	$16+1$	2.7 (3.8)	3.3 (8.3)	3.8 (1.6)	4.9 (6.4)
p_4	$1+1$	5.9 (1.0)	16.4 (1.0)	4.6 (1.0)	21.3 (1.0)
	$2+1$	5.1 (1.2)	9.3 (1.8)	5.2 (0.9)	12.6 (1.7)
	$4+1$	3.0 (2.0)	5.0 (3.3)	4.0 (1.2)	7.5 (2.9)
	$8+1$	2.5 (2.4)	3.2 (5.1)	3.2 (1.5)	5.0 (4.3)
	$12+1$	2.0 (3.0)	2.7 (6.1)	2.9 (1.6)	4.2 (5.1)
	$16+1$	1.8 (3.3)	2.6 (6.3)	2.6 (1.8)	3.6 (5.9)

of numeric growth in the original algorithm. Practical efficiency of the algorithms is heavily affected by the probabilistic nature of the factorization algorithm. There is a clear stepwise correspondence between the previous algorithms and ours, and it will be interesting to make a complete analysis and a detailed comparison of the algorithms. For other cases of Section 4.2, the expected number of candidates for exponent vectors ought to be analyzed; however, the analysis seems very difficult and is left for future research. In either case, early detection of simple terms, such as a constant term, existent in the polynomial improves the efficiency (and the probability of success).

Finally, we refer to Zippel's algorithm. The algorithm performs evaluations of a polynomial incrementally to construct its polynomial representation in a variable-by-variable manner. This means that both symbolic computation and data-parallel processing of uniform numeric data are required in the algorithm. Its parallelization and a comparison with our algorithm will be an interesting theme for future study.

References

Aho, A.V., Hopcroft, J.E., Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Ben-Or, M., Tiwari, P. (1988). A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of 20th Symposium on the Theory of Computing*, pp. 301–309, Chicago, Illinois, May 2–4.

Berlekamp, E.R. (1970). Factoring polynomials over large finite fields. *Mathematics of Computation*, **24**(111):713–735.

Blahut, R.E. (1985). *Fast Algorithms for Digital Signal Processing*. Addison-Wesley.

Brent, R.P., Gustavson, F.G., Yun, D.Y.Y. (1980). Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, **1**:259–295.

Cantor, D.G., Zassenhaus, H. (1981). A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, **36**:587–592.

Chikayama, T., Fujise, T., Sekita, D. (1994). A portable and efficient implementation of KL1. In *Programming Language Implementation and Logic Programming, Proc. 6th PLILP '94*, number 844 in LNCS, pp. 25–39. Springer-Verlag.

Grigoriev, D.Yu., Karpinski, M. (1987). The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Proc. 28th IEEE Symposium on the Foundations of Computer Science*, pp. 166–172.

Kaltofen, E., Lakshman, Y.N. (1988). Improved sparse multivariate polynomial interpolation algorithms. In Gianni, P., (ed), *Symbolic and Algebraic Computation, ISSAC '88*, number 358 in LNCS, pp. 467–474, Rome, Italy, July 4–8. Springer-Verlag.

Kaltofen, E., Lakshman, Y.N., Wiley, J-M. (1990). Modular rational sparse multivariate polynomial interpolation. In Watanabe and Nagata (1990) pp. 135–139.

Knuth, D.E. (1981). *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition.

Loos, R. (1983). Computing rational zeros of integral polynomials by p-adic expansion. *SIAM J. Computing*, **12**:286–293.

Murao, H. (1991). Vectorization of symbolic determinant calculation. *Supercomputer*, **VIII**(3):36–48.

Takahasi, H., Ishibashi, Y. (1960). A new method for the "exact calculation" by a digital computer. *Information Processing in Japan*, **1**(2):78–86. (in Japanese).

van der Waerden, B.L. (1940). *Moderne Algebra*, volume II. Springer Verlag, 2nd edition.

Villard, G. (1989). *Exact parallel solution of linear systems*, pp. 197–205. Computational Mathematics and Applications. Academic Press.

Wang, P. S. (1990). Parallel univariate polynomial factorization on shared-memory multiprocessors. In Watanabe and Nagata (1990) pp. 145–151.

Wang, P.S. (1992). Parallel univariate p-adic lifting on shared-memory multiprocessors. In Wang, P.S., (ed), *Proc. ISSAC '92*, pp. 168–176, Berkeley, CA, July 27–29.

Watanabe, S., Nagata, M., (eds). (1990). *Proc. ISSAC '90*, Tokyo, Japan, August 20–24.

Zippel, R.E. (1979). Probabilistic algorithms for sparse polynomials. In Ng, E.W., (ed), *Symbolic and Algebraic Computation, Proceedings of EUROSAM '79*, number 72 in LNCS, pp. 216–226, Marseille, France, April 5–7. Springer-Verlag.

Zippel, R. (1990). Interpolating polynomials from their values. *J. Symbolic Computation*, **9**(3):375–403.