Modular Rational Sparse Multivariate Polynomial Interpolation*

Erich Kaltofen Lakshman Y.N. John-Michael Wiley

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, 12180-3590

Abstract

The problem of interpolating multivariate polynomials whose coefficient domain is the rational numbers is considered. The effect of intermediate number growth on a speeded Ben-Or and Tiwari algorithm is studied.

Then the newly developed modular algorithm is presented. The computing times for the speeded Ben-Or and Tiwari and the modular algorithm are compared, and it is shown that the modular algorithm is markedly superior.

1 Introduction

Symbolic expressions, that is multivariate polynomials with rational coefficients, are often difficult to manipulate explicitly due to exponential growth in their size. An example is the computation of the determinant of a matrix with polynomial entries. When using straight-forward Gaussian elimination over the polynomial entry domain, it can happen that intermediate subdeterminants are very large polynomials while the final answer is an expression of modest size. In this case, however, we can obtain the value of the determinant for a specialization of the variables to arbitrary field elements by Gaussian elimination over the coefficient field without expression swell. An implicit representation of such a polynomial is that of a black box, that is a program that produces the value of the polynomial when supplied with values for the variables. In Kaltofen and Trager (1988) it is shown that this representation is applicable to several problems such as multivariate polynomial factorization.

Clearly, the process of converting a polynomial given by a black box back to the representation where the polynomial is given by a list of non-zero coefficients and corresponding terms is that of interpolation. In this paper we study the complexity of

*This material is based on work supported by the National Science Foundation under Grant No. CCR-87-05363 and Grant No. CDA-88-05910. Please direct all correspondence to the third author (wileyjm@turing.cs.rpi.edu). Appears in ISSAC '90 Proc. Internat. Symp. Symbolic Algebraic Comput., pp. 135–139, ACM Press, 1990.

the process with respect to the number of non-zero terms in this coefficient list, the so-called sparse interpolation problem. It should be noted that all classical interpolation algorithms, such as Lagrangian or Newtonian interpolation, do not account for sparsity in the answer; their running time is quadratic in the total term count, which in the multivariate case can be exponential in the non-zero monomial count even if the degree is bounded. In this paper we present an algorithm, due to Ben-Or and Tiwari (1988), for interpolating sparse multivariate polynomials where the coefficients are from a field of characteristic zero. Traditionally computations involving polynomials of this type have suffered from intermediate expression swell. It was Brown (1971) who first applied modular techniques to Euclids algorithm for polynomial greatest common divisor computations, the use of which greatly improves the running time of the algorithm (see [Brown, 1971]). Here we report how these techniques can be applied to the problem of interpolating sparse multivariate polynomials.

Notation

Let $P(x_1, \ldots, x_n) = c_1 m_1 + c_2 m_2 + \ldots + c_t m_t$ be the polynomial to be interpolated. The $m_i = x_1^{e_{i,1}} \ldots x_n^{e_{i,n}}$ are distinct terms and the c_i are the corresponding non-zero coefficients; t is the number of terms in P. Let $b_i = p_1^{e_{i,1}} \ldots p_n^{e_{i,n}}$ denote the value of the monomial m_i at $(p_1, \ldots, p_n)$ where p_i is the i-th prime number. Clearly, different terms evaluate to different values under this evaluation. Let

$$a_i = P(p_1^i, p_2^i, \ldots, p_n^i), 0 \leq i \leq \tau - 1$$

where $\tau \geq t$. We have $a_i = \sum_{j=1}^{\tau} c_j b_j^i$.

2 The Ben-Or and Tiwari Interpolation Algorithm

The algorithm needs as input an upper bound $\tau \geq t$ on the number of terms in P. The algorithm proceeds in two stages. The monomial values m_i are determined first by the use of an auxiliary polynomial $\Lambda(z)$. Once the m_i are known, the coefficients c_i can be obtained easily. The polynomial $\Lambda(z)$ is constructed as follows. Let

$$\Lambda(z) = \prod_{i=1}^{t} (z - m_i) = z^t + \lambda_{t-1} z^{t-1} + \ldots + \lambda_1 z + \lambda_0.$$

Consider the sum

$$\sum_{i=1}^{t} c_i m_i^j \Lambda(m_i) = \sum_{k=0}^{t-1} \lambda_k (c_1 m_1^{k+j} + c_2 m_2^{k+j} + \ldots + c_t m_t^{k+j}) + (c_1 m_1^{t+j} + c_2 m_2^{t+j} + \ldots + c_t m_t^{t+j})$$

for all j, $0 \leq j \leq t-1$. Since $\Lambda(m_i) = 0$, we have

$$a_j \lambda_0 + a_{j+1} \lambda_1 + \ldots + a_{j+t-1} \lambda_{t-1} + a_{j+t} = 0, \qquad 0 \leq j \leq t-1.$$

We now have the Toeplitz system $A_t\vec{\lambda_t} = \vec{a_t}$ where

$$A_i = \begin{pmatrix} a_{t-1} & a_t & \dots & a_{t-2+i} \\ a_{t-2} & a_{t-1} & \dots & a_{t-3+i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{t-i} & a_{t-i+1} & \dots & a_{t-1} \end{pmatrix}, \qquad \vec{\lambda_i} = \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{i-1} \end{pmatrix}, \qquad \vec{a_i} = - \begin{pmatrix} a_{t-1+i} \\ a_{t-2+i} \\ \vdots \\ a_t \end{pmatrix}.$$

This system is non-singular as can be seen from the factorization

$$A_t = \begin{pmatrix} b_1^{t-1} & b_2^{t-1} & \dots & b_t^{t-1} \\ b_1^{t-2} & b_2^{t-2} & \dots & b_t^{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} c_1 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & c_t \end{pmatrix} \begin{pmatrix} 1 & b_1 & \dots & b_1^{t-1} \\ 1 & b_2 & \dots & b_2^{t-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & b_t & \dots & b_t^{t-1} \end{pmatrix}.$$

Since the b_i are distinct, the two Vandermonde matrices are non-singular and as no c_i is zero, the diagonal matrix is nonsingular too. If the input value of the upper bound τ is greater than t, then the coefficients c_k, for $k > t$, can be regarded as zero and the resulting system A_τ will be singular.

The roots of the polynomial $\Lambda(z)$ give the b_i and by choosing the first t evaluations of P, we get the following transposed Vandermonde system of equations $V\vec{c} = \vec{a}$ for the coefficients of P, where

$$V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ b_1 & b_2 & \dots & b_t \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{t-1} & b_2^{t-1} & \dots & b_t^{t-1} \end{pmatrix} \qquad \vec{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{pmatrix}, \qquad \vec{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix} \tag{1}$$

We now state the complete interpolation algorithm.

Algorithm *Polynomial Interpolation (by Ben-Or and Tiwari)*

Input: A black box for evaluating a multivariate polynomial $P(x_1, \dots, x_n)$ and an upper bound τ on the number of terms in P.

Output: The polynomial $P = \sum_{i=1}^{t} c_i m_i$ where t, c_i, m_i all denote the same quantities as before.

Step 1: For i from 0 to $2\tau - 1$ do $a_i := P(p_1^i, p_2^i, \dots, p_n^i)$.

Step 2: Find the rank t of the matrix A_τ

Step 3: Solve the Toeplitz system $A_t\vec{\lambda_t} = \vec{a_t}$ to recover the auxiliary polynomial $\Lambda(z)$.

Step 4: Find the integer roots of $\Lambda(z)$ to get the b_i. Compute the monomial m_i from b_i by repeatedly dividing b_i by $p_1, \dots, p_n$.

Step 5: Find the coefficients c_i by solving the transposed Vandermonde system $V\vec{c} = \vec{a}$ described earlier.

3 The Modular Rational Interpolation Algorithm

The Ben-Or and Tiwari algorithm was implemented with the following improvements.

Recovering the auxiliary polynomial $\Lambda(z)$ in Steps 2 and 3 is accomplished by viewing the evaluation points as a linear sequence. Then $\Lambda(z)$ is the feedback connection polynomial which generates the sequence $a_0, a_1, \ldots, a_{2\tau-1}$ [Blahut, 1983]. Thus the Berlekamp-Massey algorithm can be used to recover $\Lambda(z)$.

The Berlekamp-Massey algorithm recursively constructs a minimum length feedback connection polynomial $\Lambda(z)$ of length at most τ for the sequence $a_0, a_1, \ldots, a_{2\tau-1}$. It does this by calculating a feedback connection polynomial $\Lambda_i(z)$ at each step i, which satisfies the subsequence $a_0, a_1, \ldots, a_{2i-1}$, where $\Lambda_0(z) = 1$. Each of the *intermediate feedback connection polynomials*, $\Lambda_{2k-1}(z)$ for all $k, 1 \leq k \leq \tau$, must be unique [Massey, 1969]. Thus the intermediate feedback connection polynomials satisfy the Toeplitz systems $\bar{A}_i \vec{\lambda_i} = \vec{a_i}$ where

$$\bar{A}_i = \begin{pmatrix} a_0 & a_1 & \ldots & a_{i-1} \\ a_1 & a_2 & \ldots & a_i \\ \vdots & \vdots & \ddots & \vdots \\ a_{i-1} & a_i & \ldots & a_{2i-2} \end{pmatrix}, \qquad \vec{\lambda_i} = \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_i \end{pmatrix}, \qquad \vec{a_i} = - \begin{pmatrix} a_i \\ a_{i+1} \\ \vdots \\ a_{2i-1} \end{pmatrix},$$

and λ_k is the kth coefficient of Λ_{2i-1}.

It was observed during trial runs that the coefficients in the Berlekamp-Massey algorithm were extremely large. For example, in our test case involving three variables (see table in section 4) and $\Lambda(z)$ having a degree bound of 250, $\Lambda(0)$ had approximately 1400 decimal digits.

To control the coefficient growth, all computations were performed modulo p^k where p is a prime and p^k is *sufficiently large*. The prime power p^k is *sufficiently large* if each m_i is less than p^k. Choosing $p^k > m_i$ implies that the roots of $\Lambda(z)$ modulo p^k are the the m_i. Thus the modular image of $\Lambda(z)$ is sufficient for recovering the m_i.

Finding the integer roots of $\Lambda(z)$ in Step 4 is accomplished using a randomized Cantor-Zassenhaus algorithm (see [Cantor and Zassenhaus, 1981]) to factor $\Lambda(z)$ modulo a prime p. Then the true roots are reconstructed from their modular images by Hensel lifting. Furthermore, we only need a modular image of $\Lambda(z)$. The transposed Vandermonde system in step 5 can be solved using the method of Zippel (1990).

We now state the modular version of the Ben-Or and Tiwari interpolation algorithm.

Algorithm *Modular Rational Interpolation*

Input: A black box for evaluating a multivariate polynomial $P(x_1, \ldots, x_n)$, an upper bound τ on the number of terms in P, a degree bound d for the polynomial, and bound C for the numerator and denominator of the coefficients appearing in P.

Output: The polynomial $P = \sum_{i=1}^{t} c_i m_i$, or *failure*.

Step 1 (Evaluation): For i from 0 to $2\tau - 1$ do $a_i := P(p_1^i, p_2^i, \ldots, p_n^i)$.

Step 2 (Choosing a prime p, and k): Choose a prime p, so that p satisfies the conditions of Lemma 1. Then choose k so that $p^k > \max\{p_n^d, 2C^2\}$.

Step 3 (Computing the feedback connection polynomial $\Lambda(z) \bmod p^k$): Use a modular Berlekamp-Massey algorithm to compute $\Lambda(z)$ mod p^k from the a_i mod p^k.

Step 4 (Finding the Integer Roots of $\Lambda(z)$): Find the integer roots of $\Lambda(z)$ by finding the roots of $\Lambda(z)$ mod p, and then lifting to the true roots of $\Lambda(z)$ mod p^k.

Step 5: Find the mod p^k images of the coefficients c_i by solving the transposed Vandermonde system $V\vec{c} = \vec{a}$ using a modular realization of Zippel's algorithm (1990).

Step 6: Recover the rational coefficients from their images modulo p^k using continued fraction recovery [Wang *et al.*, 1982].

Step 7: Check if $a_i = \bar{P}(p_1^i, p_2^i, \ldots, p_n^i)$ where $\bar{P}$ is the found polynomial.

The algorithm will fail if the prime p is not *lucky*. A prime p is *lucky* if the feedback connection polynomial produced by the modular version of the Berlekamp-Massey algorithm has the same degree as the the true (rational) feedback connection polynomial, and if $\Lambda(z)$ mod p remains square free.

If the prime p does not divide the numerator or the denominator of the leading coefficients of any intermediate feedback connection polynomials computed during the rational Berlekamp-Massey algorithm then the modular algorithm produces a feedback connection polynomial of the same degree. The following lemma characterizes the suitable primes for the Berlekamp-Massey algorithm.

Lemma 1 *Let p be a prime chosen uniformly randomly in the range* $\max\{8, p_n\} < p < D$ *where* $D = 14dt^3 \max(1, \log n) + 8t^2 \log C$. *Then the probability that p does not divide the numerator or the denominator of the leading coefficients of any of the feedback connection polynomials computed by the rational Berlekamp-Massey algorithm is at least 3/4.*

Proof. Let $a = tCp_n^{2dt}$. By the definition of the a_i, $a \geq a_i$, for all $i, 0 \leq i \leq 2t-1$.

Since the intermediate feedback connection polynomials are solutions to linear systems of the form $\bar{A}_i \hat{\lambda}_i = \hat{a}_i$, then the numerator and denominator of the leading coefficient of for any of these intermediate feedback connection polynomials are bounded by $(ia^2)^{1/2}$. The product of all the leading coefficients is thus bounded by the value

$$E = t^{4t^2} C^{2t^2} p_n^{3dt3}.$$

Consider the r primes $p_k, p_{k+1}, \ldots, p_{k+r}$, where k is the smallest integer such that $p_k \geq \max\{8, p_n\}$, and $r = 7dt^3 \log n + 4t^2 \log C$. Now for any r/4 these primes,

$$\prod_{j=1}^{r/4} p_j > 2^{3r/4} > E.$$

Hence at most $r/4$ of these primes divide the product of the numerators and denominators of the leading coefficients. Thus if a prime p is chosen uniformly randomly in the given range, the probability that p does not divide E is at least 3/4. ■

Once $\Lambda(z) \bmod p^k$ has been found, the prime p can be used in step 5 for finding the roots of $\Lambda(z)$. The roots, modulo p, can be lifted to the true roots of $\Lambda(z) \bmod p^k$, which are the true roots of $\Lambda(z)$. The prime chosen is large enough to insure a high probability for success during the root finding stage [Kaltofen *et al.*, 1989].

Large intermediate numbers were noticed also in the final step of the interpolation algorithm, that of recovering the rational coefficients of the interpolating polynomial by solving a transposed Vandermonde system. A modular solver was used and the rational coefficients were recovered using a continued fraction recovery method[Wang *et al.*, 1982].

4 Implementation

The algorithm was coded in AKCL (Austin Kyoto Common Lisp) running on Sun4 (Unix Sun OS 4.0.3). It takes as input the number of variables n and the term bound τ.

It first uses the input values to generate a random polynomial (Lisp function) to be used as the black box. It uses the given black box to generate the sequence $a_0, a_1, \ldots, a_{2\tau-1}$. Then it chooses a random prime, p so that $p_n < p \leq \tau^3$. An integer k is chosen so that $p^k > p_n^d$. It is assumed that $p_n^d > 2C^2$ (C is the bound on the numerator and denominator of the coefficients), this way the true coefficients can be recovered in step 6. Once the modulus p^k has been chosen the program steps 3, 4, and 5 are then performed. If at any time during these computations the program tries to invert a divisor of zero, or if the feedback connection polynomial is not square free modulo p, the program returns to step 2.

The following table summarizes the results of the experiments. The values for the run times are the average of all the runs completed for polynomials of the given size. For most of the cases, this involves at least twenty different polynomials. For the larger case (60 to 250 terms) fewer computations were used.

	Fast Ben-Or and Tiwari		Modular Algorithm	
Term Bound	3 Vars	4 Vars	3 Vars	4 Vars
5	20.8s	42.0s	3.0s	5.7s
10	1622.0s	3807.4s	9.9s	17.2s
15	15542.3s	57550.0s	20.4s	32.5s
20	95953.4s	?	36.2s	58.6s
30	?	?	92.3s	143.8s
60	?	?	602.3s	1051.2s
100	?	?	2627.1s	4690.3s
250	?	?	57056.6s	131652.6s

? - Computation did not complete

Conclusion

In attempting to implement Ben-Or and Tiwari's algorithm for sparse multivariate polynomial interpolation where the coefficient domain was the rational numbers, one immediately encounters the problem of intermediate coefficient growth. The coefficient growth causes the running times for problems of a reasonable size to grow unreasonably large.

The modular algorithm avoids this problem by mapping the problem into a domain in which the solutions can be more easily computed. The modulus is chosen so as to preserve the roots of the error locator polynomial, and so that the true coefficients can be recovered using continued fractions.

References

[Ben-Or and Tiwari, 1988] M. Ben-Or and P. Tiwari. "A deterministic algorithm for sparse multivariate polynomial interpolation", In *Proc. 20th annual ACM Symp. Theory Comp.*, pages 301–309, 1988.

[Blahut, 1983] R.E. Blahut. *Theory and Practice of Error Correcting Codes.* Addison and Wesley, Reading, Mass., 1983.

[Brown, 1971] W.S. Brown. "On Euclid's algorithm and the computation of polynomial greatest common divisors", *Journal of the ACM*, 18(4):478–504, October 1971.

[Cantor and Zassenhaus, 1981] David G. Cantor and Hans Zassenhaus. "A new algorithm for factoring polynomials over finite fields", *Mathematics of Computation*, 36(154):587–592, April 1981.

[Kaltofen and Trager, 1988] E. Kaltofen and B. Trager. "Computing with polynomials given by black boxes for their evaluation: Greatest common divisors, factorization, seperation of numerators and denominators", In *Proc. 29th Annual Symp. Foundations of Comp. Science*, pages 296–305, 1988. Also to Appear in *J. Symbolic Computation.*

[Kaltofen and Yagatil, 1988] E. Kaltofen and L. Yagatil. "Improved sparse multivariate poynomial interpolation algorithms", *Proc. ISSAC '88, Springer Lect. Notes Computer Science*, 358:467–474, 1988.

[Kaltofen *et al.*, 1989] E. Kaltofen, Y.N. Lakshman, and J.M. Wiley. "Efficient sparse multivariate poynomial interpolation algorithms", Manuscript, December 1989.

[Massey, 1969] J. Massey. "Shift-register synthesis and BCH coding", *IEEE Trans. Inform. Theory*, IT-15:122–127, 1969.

[Wang *et al.*, 1982] P.S. Wang, M.J.T. Guy, and J.H. Davenport. "P-adic reconstruction of rational numbers", *SIGSAM Bulletin*, 16(2):2–3, May 1982.