Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's Algorithm*

Erich Kaltofen[1] Wen-shin Lee[1] Austin A. Lobo[2]
[1]Department of Mathematics, North Carolina State University, Raleigh, North Carolina 27695-8205
{kaltofen,wlee1}@math.ncsu.edu; http://www.kaltofen.net; http://www.wen-shin.com
[2]Dept. of Mathematics and Computer Science, Washington College, Chestertown, Maryland 21620
austin.lobo@washcoll.edu; http://www.austin.lobo.washcoll.edu/

1. INTRODUCTION

Interpolation algorithms whose computational complexities are sensitive to the sparsity of their target polynomials are one of the major contributions of computer algebra to computer science and mathematics. The first such result was obtained by Richard Zippel in 1979 [21]. Zippel's algorithm is efficient in the multivariate case when the polynomial to be interpolated has much fewer non-zero monomials than a variable by variable Lagrange or Newton interpolation algorithm would attempt to find. Zippel's algorithm is still a variable by variable interpolation method and requires randomization. In 1988 Michael Ben-Or and Prasoon Tiwari gave a different algorithm [1] that is based the Berlekamp/Massey algorithm from coding theory. Their algorithm is not variable by variable and works equally well for sparse univariate polynomials. In its original form, it does not require randomization, but for its correctness it must be given an upper bound for the number of terms in the target polynomial.

Since then, both approaches have been generalized and improved. The Vandermonde techniques of Ben-Or and Tiwari can be applied to Zippel's algorithm [22, 11]. Lakshman Y. N., B. David Saunders, and Dima Yu. Grigoriev extended the Ben-Or/Tiwari approach to sparsity with respect to non-standard polynomial bases, such as Chebyshev bases and shifted bases [14, 8, 15]. For polynomials over small finite fields both Zippel's and Ben-Or's and Tiwari's algorithms require modification [7; 20, and the references given there]. Both Zippel's and Ben-Or's and Tiwari's interpolation algorithms have been implemented by several authors on both single and multi-processor computers, among them [10, 17, 4].

*This material is based on work supported in part by the National Science Foundation under Grant Nos. CCR-9712267 and DMS-9977392 (Kaltofen and Lee) and Grant No. INT-9726763 (Kaltofen and Lobo). Version 4/24/2001 (17:8).

ISSAC 2000, St. Andrews, Scotland
©2000 ACM 1-58113-218-2/ 00/ 0008 $5.00

In this paper we revisit the original two algorithms. Zippel's algorithm has a shortcoming over Ben-Or's and Tiwari's in that it proceeds one variable at a time, and each new variable is interpolated densely. Ben-Or's and Tiwari's algorithm requires knowledge of an upper bound of the number of terms, and probes the target polynomial twice as many times. Furthermore, when implementing Ben-Or's and Tiwari's algorithm [10] in a modular fashion, in the multivariate case to our knowledge the modulus must be large enough for recovery of all terms evaluated at prime numbers. For univariate polynomials there are special tricks that reduce the size of the modulus. We note that implementation of Ben-Or's and Tiwari's algorithm with rational number arithmetic causes extreme intermediate expression swell. On the other hand, in Zippel's algorithm the modulus only needs to capture the coefficients, and the number of residues must be large enough for randomization. Furthermore, it adaptively determines the term structure of the polynomial.

The main innovations in our new algorithm are two-fold. First, we show that Ben-Or's and Tiwari's algorithm can be made to adapt to the term number as well. The key fact comes as a theorem: when choosing random evaluation points, a zero discrepancy in the Berlekamp/Massey substep indicates that the term bound has been reached, that with high probability. We call this phenomenon *early termination*, borrowed from our idea in the setting of the Wiedemann algorithm [19]. For multivariate polynomials, the required size of the modulus would still remain quite large. Therefore, we revert back to Zippel's algorithm, but now implement each univariate polynomial interpolation as the early termination version of the Ben-Or/Tiwari algorithm. Thus is the genesis of the hybrid Ben-Or/Tiwari/Zippel algorithm.

Zippel's algorithm, the early termination version of Ben-Or's and Tiwari's algorithm and the hybrid algorithm are all randomized in the Monte Carlo sense, i.e, their results are correct with high probability. We do not know how to verify the answer. Therefore, in our implementation, we adopt the strategy of putting additional partial verification computations into our procedure. For example, early termination is only triggered after encountering a series of discrepancies that are repeatedly zero. The length of the series is a threshold that is given as an optional argument to the procedure. Although we have proven that the early termination strat-

egy is probabilistically correct for threshold one, we note that higher thresholds weed out bad random choices from sets that are much smaller than the early termination theorem would require. The algorithm then becomes a heuristic that can interpolate polynomials of a size at the very edge of what current software and hardware can reach.

Finally, we may perform univariate Newton interpolation at the same points that the univariate Ben-Or/Tiwari algorithm uses, thus requiring no additional probes of the target polynomial. This leads us to implement a race between the two algorithms, and "early-terminate" the Newton interpolation part as soon as the interpolating polynomial has stabilized, again for threshold many new points. By simultaneous use of Newton interpolation one not only obtains a dense univariate result from fewer evaluations, but one may also cross-check a sparse Ben-Or/Tiwari answer for degree consistency with the partial Newton interpolant.

We have implemented our ideas in Maple V.5 as a black box polynomial sparse interpolation over the integers modulo a prime number. We use term pruning via homogenization as described in [4]. We present the performance of our implementation on a series of benchmark polynomials by comparing the number of black box probes required to those in the previous algorithms. Clearly, there is a tradeoff between the additional arithmetic operations introduced by using two univariate interpolation algorithms inside Zippel's and the savings of probes of the target polynomial. We intend our algorithm for polynomials produced by the calculus of black box polynomial [13, 4]. Nonetheless, the usage of early termination Ben-Or/Tiwari inside Zippel is more efficient for many sparse inputs.

2. THE BEN-OR/TIWARI INTERPOLATION ALGORITHM AND ITS EARLY TERMINATION

2.1 The Berlekamp/Massey algorithm

For later reference, we shall give the Berlekamp/Massey algorithm [16]. The algorithm processes a stream of elements $a_0, a_1, \ldots \in \mathbb{K}$, where $\mathbb{K}$ is an arbitrary field. If the sequence has a linear generator $\Lambda(z) = z^t - \lambda_{t-1}z^{t-1} - \cdots - \lambda_0$ of degree t, meaning that for all $j \geq 0$ we have $a_{t+j} = \lambda_{t-1}a_{t+j-1} + \cdots + \lambda_0 a_j$, the algorithm will compute it after processing $2t$ elements from the stream. The stream is unbounded, however, and the algorithm can update the current guess for the linear generator appropriately whenever the next stream element a_i does not fit the current linear recursion. In that case a non-zero discrepancy is detected. There are two updates, one where the generator jumps in degree (Step 3 below) and one where the lower order coefficients of the generator get adjusted (Step 4 below). Note that the algorithm computes the reverse of the generator polynomial.

The Berlekamp/Massey algorithm

Input: $a_0, a_1, \ldots \in \mathbb{K}$

1. (Initialization.)
   $\Lambda_0 \leftarrow 1$; $B_0 \leftarrow 0$; $L_0 \leftarrow 0$; $\Delta \leftarrow 1$;
   For $r = 1, 2, \ldots$ ***Do***

2. (First, we compute the *discrepancy* Δ_r, assuming
   $$\Lambda_{r-1}(z) = \lambda_0 z^s + \lambda_1 z^{s-1} + \cdots + \lambda_s,$$
   where $s = \deg \Lambda_{r-1}$ and $\lambda_0, \ldots, \lambda_s \in \mathbb{K}$ with $\lambda_0 \neq 0$; note that we always have $\lambda_s = 1$.)
   $\Delta_r \leftarrow \lambda_s a_{r-1} + \lambda_{s-1}a_{r-2} + \cdots + \lambda_0 a_{r-s-1}$;
   If $\Delta_r = 0$ *then* $\Lambda_r \leftarrow \Lambda_{r-1}$; $B_r \leftarrow zB_{r-1}$; $L_r \leftarrow L_{r-1}$;

3. *If* $\Delta_r \neq 0$ *and* $2L_{r-1} < r$ *then*
   $B_r \leftarrow \Lambda_{r-1}$; $\Lambda_r \leftarrow \Lambda_{r-1} - (\Delta_r/\Delta) \cdot zB_{r-1}$;
   $L_r \leftarrow r - L_{r-1}$; $\Delta \leftarrow \Delta_r$;

4. *If* $\Delta_r \neq 0$ *and* $2L_{r-1} \geq r$ *then*
   $\Lambda_r \leftarrow \Lambda_{r-1} - (\Delta_r/\Delta) \cdot zB_{r-1}$; $B_r \leftarrow zB_{r-1}$;
   $L_r \leftarrow L_{r-1}$;

End For;
End.

2.2 The Ben-Or/Tiwari algorithm

Let f be a multivariate polynomial, m_j its distinct terms, t the number of terms, and c_j the corresponding non-zero coefficients:

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{t} c_j x_1^{e_{j,1}} \cdots x_n^{e_{j,n}} = \sum_{j=1}^{t} c_j m_j \quad c_j \neq 0.$$

Let $b_j = p_1^{e_{j,1}} \cdots p_n^{e_{j,n}}$, where p_i are values from the coefficient domain to be specified later, and let

$$a_i = f(p_1^i, \ldots, p_n^i) = \sum_{j=1}^{t} c_j b_j^i.$$

Now define an auxiliary polynomial $\Lambda(z)$ as following:

$$\Lambda(z) = \prod_{j=1}^{t} (z - b_j) = z^t + \lambda_{t-1}z^{t-1} + \cdots + \lambda_0.$$

THEOREM 1. *For $i \geq 0$, $a_i = f(p_1^i, \ldots, p_n^i)$, the sequence of $\{a_i\}_{i\geq 0}$ is linearly generated by the polynomial $\Lambda(z)$. Furthermore, $\Lambda(z)$ is the minimal polynomial of $\{a_i\}_{i\geq 0}$* [1].

In subsection 2.1 we give the Berlekamp/Massey algorithm that computes $\Lambda(z)$ from $\{a_i\}_{i\geq 0}$. We obtain the evaluated terms b_j by finding the roots of $\Lambda(z)$. We then need to recover each $m_j = x_1^{e_{j,1}} \cdots x_n^{e_{j,n}}$ from b_j. If the values p_i are consecutive prime numbers and evaluation is done over a coefficient field of characteristic zero, we can do this by repeatedly dividing b_j by $p_1, \ldots, p_n$. Finally, we need to determine all the corresponding coefficients c_j of m_j in order to complete the interpolation of f. We do this by solving a $t \times t$ linear system for the coefficients, $a_i = \sum_{j=1}^{t} c_j b_j^i$ with $0 \leq i \leq t-1$. This turns out to be a transposed Vandermonde system:

$$\begin{bmatrix} 1 & 1 & \ldots & 1 \\ b_1 & b_2 & \ldots & b_t \\ b_1^2 & b_2^2 & \ldots & b_t^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{t-1} & b_2^{t-1} & \ldots & b_t^{t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{t-1} \end{bmatrix}. \qquad (1)$$

Efficient algorithms for solving transposed Vandermonde systems can be found in [11, 22].

The Ben-Or/Tiwari interpolation algorithm

Input: f: *a multivariate black box polynomial.*
τ: $\tau \geq t$, *t is the number of terms in f.*
Output: c_j *and* m_j: $f = \sum_{j=1}^{t} c_j m_j$

1. (The Berlekamp/Massey algorithm)
   $a_i = f(p_1^i, \ldots, p_n^i), 0 \leq i \leq \tau$. *where p_i is the i-th prime.*
   Compute $\Lambda(z)$ *from* $\{a_i\}_{\tau \geq i \geq 0}$.
2. (Determine m_j)
   Find all t distinct roots of $\Lambda(z)$, b_j.
   Determine all m_j: *repeatedly divide every* b_j *by* $p_1, \ldots, p_n$.
3. (Compute c_j as described above.)
   Solve a transposed Vandermonde system.
   End.

2.3 Early termination

Both the Ben-Or/Tiwari [1] and the Kaltofen *et al.* [10] algorithms need to know the number of terms, t, or an upper bound τ, $\tau \geq t$. Otherwise, we can guess t, compute a sparse candidate polynomial g for f, and if there is no failure in finding the roots of Λ (which happens most of the times), compare g and f at an additional random point. If the values are different, one can double the guess for t. The algorithm is randomized in the Monte Carlo sense.

Based on the strategy of "early termination," we present a more efficient probabilistic approach, which requires a single interpolation run. The idea is simple: pick random point coordinates p_k for the Ben-Or/Tiwari algorithm and show that with high probability the embedded Berlekamp/Massey algorithm of subsection 2.1 does not encounter a zero discrepancy Δ (in step 2 for the case that $2L < r$,* i.e., by which would be divided in step 3 if the discrepancy were non-zero) until $r > 2t$. However, we cannot claim this is generally true. For a polynomial $f(x) = \sum_{j=1}^{t} c_j x^{e_j}$ that satisfies $f(p^0) = a_0 = c_1 + \cdots + c_t = 0$, the first discrepancy $\Delta_1 = 0$. Yet, by shifting the sequence by 1 element, the early termination property can be proved (with high probability).

We first prove that for symbolic values, namely the variables $x_1, \ldots, x_n$, the first zero discrepancy (for $2L < r$) appears at $r = 2t + 1$. Let $\beta_j = x_1^{e_{j,1}} \cdots x_n^{e_{j,n}}$ be the j-th non-zero term in f, and let $\alpha_i = f(x_1^i, \ldots, x_n^i)$. We have

$$\mathcal{A}_i = \begin{bmatrix} \alpha_{2i-1} & \alpha_{2i-2} & \ldots & \alpha_i \\ \alpha_{2i-2} & \alpha_{2i-3} & \ldots & \alpha_{i-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_i & \alpha_{i-1} & \ldots & \alpha_1 \end{bmatrix} \tag{2}$$

$$= \mathcal{B}_i C_t \bar{\mathcal{B}}_i^{\mathrm{Tr}}, \tag{3}$$

where

$$\mathcal{B}_i = \begin{bmatrix} \beta_1^{i-1} & \beta_2^{i-1} & \ldots & \beta_t^{i-1} \\ \beta_1^{i-2} & \beta_2^{i-2} & \ldots & \beta_t^{i-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{bmatrix}, \; C_t = \begin{bmatrix} c_1 & 0 & \ldots & 0 \\ 0 & c_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & c_t \end{bmatrix}$$

and

$$\bar{\mathcal{B}}_i = \begin{bmatrix} \beta_1^i & \beta_2^i & \ldots & \beta_t^i \\ \beta_1^{i-1} & \beta_2^{i-1} & \ldots & \beta_t^{i-1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1 & \beta_2 & \ldots & \beta_t \end{bmatrix}.$$

The singularity of the Hankel matrix $\mathcal{A}_i$ (2) is directly related to the vanishing of discrepancies when computing a linear generator of $\alpha_1, \alpha_2, \ldots$ by the Berlekamp/Massey algorithm. The argument makes use of the interpretation of a Berlekamp/Massey algorithm as the extended Euclidean algorithm on the polynomials $F_{-1} = X^N$ and $F_0 = \alpha_1 X^{N-1} + \alpha_2 X^{N-2} + \cdots$ [5] combined with the fundamental theorem on subresultants [2]. Here N is the number of elements that are considered for determining the linear generator. Dornstetter shows that Δ_r in Step 3 of the Berlekamp/Massey algorithm of subsection 2.1 is the leading coefficient in a remainder, F_i, in a polynomial remainder sequence (PRS) of F_{-1} and F_0. The discrepancies in Step 4 are the trailing coefficients in F_i. Step 2 processes both trailing coefficients that are zero and the search for the non-zero leading coefficient of the next remainder. The former can be diagnosed by the shift-register length $2L \geq r$. The remainder polynomials F_i in the PRS, which yields the polynomials B_r and Λ_r at $2L = r$ as the reverse polynomials of the consecutive Bezout coefficients T_{i-1} and T_i in the extended Euclidean scheme $F_i = S_i F_{-1} + T_i F_0 \equiv T_i F_0 \pmod{X^N}$, are adjusted by non-zero scalar multipliers, namely $-\Delta_r/\Delta$ of Step 3. Dornstetter's analysis yields the following fact.

FACT 1. *If the PRS is normal, i.e.,* $\deg(F_i) = \deg(F_{i-1}) - 1 = N - i - 1$, *where* $i \geq 0$, *then* $\Delta_r \neq 0$ *whenever* $2L < r$ *and* $L < t$, *where t is the degree of the linear generator.*

Appealing now to the fundamental theorem of subresultants, the PRS is normal if and only if the leading coefficient of the $N - i - 1$'st subresultant of F_{-1} and F_0 does not vanish. By definition [2], this is the determinant of a $(2i+1) \times (2i+1)$ matrix shown in figure 1.

$$\det \begin{bmatrix} 1 & 0 & \cdots & & 0 & & & 0 \\ & \ddots & & & & \ddots & & \vdots \\ & & 1 & & & & & 0 \\ & & & \ddots & & & & \vdots \\ 0 & & \cdots & & 1 & 0 & \cdots & 0 \\ \alpha_1 & & & \cdots & \alpha_i & \alpha_{i+1} & & \alpha_{2i+1} \\ & \ddots & & & & & & \\ & & \alpha_1 & & & \vdots & \ddots & \vdots \\ & & & & \ddots & & & \\ 0 & & & & 0 & \alpha_1 & \cdots & \alpha_{i+1} \end{bmatrix} = \pm \det(\mathcal{A}_{i+1}).$$

Figure 1: Subresultant coefficient

The early termination strategy is correct (for symbolic evaluations) if the determinant of $\mathcal{A}_i$ is non-zero. This is our next theorem.

*Clearly, the shift register length is stored in a single integer variable without a subscript.

THEOREM 2. *The determinant of $\mathcal{A}_i$ is non-zero.*

Proof: Let $M_{J,K}$ be the determinant of the submatrix of M consisting of the rows listed in the set J and the columns listed in the set K. The Binet-Cauchy formula [6] states for a matrix product AB that

$$(AB)_{J,L} = \sum_{1\le k_1<k_2<\cdots<k_i\le n} A_{J,\{k_1,\ldots,k_i\}} B_{\{k_1,\ldots,k_i\},L}, \quad (4)$$

where n is the number of columns of A and J and L are sets of row and column indices with i elements each. Applying (4) to (3) with $I = \{1,\ldots,i\}$ we have

$$\begin{aligned}
\det(\mathcal{A}_i) &= (\mathcal{B}_i C_t \bar{\mathcal{B}}_i^{\mathrm{Tr}})_{I,I} \\
&= \sum_J \sum_K (\mathcal{B}_i)_{I,J} (C_t)_{J,K} (\bar{\mathcal{B}}_i^{\mathrm{Tr}})_{K,I} \\
&= \sum_J (\mathcal{B}_i)_{I,J} (C_t)_{J,J} (\bar{\mathcal{B}}_i^{\mathrm{Tr}})_{J,I} \\
&= \sum_{J=\{j_1,\ldots,j_i\}} c_{j_1}\cdots c_{j_i}\, \beta_{j_1}\beta_{j_2}\ldots\beta_{j_i} \\
&\qquad \cdot \det\begin{pmatrix} \beta_{j_1}^{i-1} & \beta_{j_2}^{i-1} & \ldots & \beta_{j_i}^{i-1} \\ \beta_{j_1}^{i-2} & \beta_{j_2}^{i-2} & \ldots & \beta_{j_i}^{i-2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{pmatrix}^2 \\
&= \sum_{J=\{j_1,\ldots,j_i\}} c_{j_1}\cdots c_{j_i}\, \beta_{j_1}\beta_{j_2}\ldots\beta_{j_i} \\
&\qquad \cdot \prod_{1\le v<u\le i} (\beta_{j_u} - \beta_{j_v})^2. \qquad (5)
\end{aligned}$$

Now let the terms $\beta_1 \succ \beta_2 \succ \cdots \succ \beta_t$ be ordered lexicographically. Then the summand

$$c_1\cdots c_i\, \beta_1\beta_2\ldots\beta_i \prod_{1\le v<u\le i} (\beta_v - \beta_u)^2$$

has the term

$$\beta_1^{2i-1}\beta_2^{2i-3}\cdots\beta_i$$

which occurs nowhere else,† hence $\det(\mathcal{A}_i)$ does not vanish. ⊠

We make the transition from symbolic point coordinates $x_1,\ldots,x_n$ to random field elements $p_1,\ldots,p_n$ in the customary fashion via the the Schwartz-Zippel lemma [21, 18] (see also [3]).

THEOREM 3. *If $p_1,\ldots,p_n$ are chosen randomly and uniformly from a subset S of the domain of values, which is assumed to be an integral domain, then for the sequence $a_i = f(p_1^i,\ldots,p_n^i)$ the Berlekamp/Massey algorithms encounters $\Delta = 0$ and $2L < r$ the first time for $r = 2t+1$ with probability no less than*

$$1 - \frac{t(t+1)(2t+1)\deg(f)}{6\cdot\#(S)},$$

where $\#(S)$ is the number of elements in S.

†In this argument we make use of the shift by 1 element. We do not know if shifting is actually needed if one were to exclude the first discrepancy from the termination test.

Proof: By (5) we obtain $\deg(\det\mathcal{A}_i) \le i^2\deg(f)$. We have to avoid a zero of the product $\prod_{i=1}^t \det\mathcal{A}_i$, whose degree is no more than $t(t+1)(2t+1)\deg(f)/6$. The estimate of the probability follows from Lemma 1 in [18]. ⊠

We conclude the probabilistic analysis with three remarks.

Remark 1: If the coefficient field is a subfield of the real numbers and $c_i > 0$ for all i, no randomization is necessary. The following argument is standard for the least squares problem with a weighted inner product:

$$\begin{aligned}
B_i C_t B_i^{\mathrm{Tr}} y = 0 &\Longrightarrow y^{\mathrm{Tr}} B_i C_t B_i^{\mathrm{Tr}} y = 0 \\
&\Longrightarrow (B_i^{\mathrm{Tr}} y)^{\mathrm{Tr}} C_t (B_i^{\mathrm{Tr}} y) = 0 \\
&\Longrightarrow B_i^{\mathrm{Tr}} y = 0,
\end{aligned}$$

because $0 = z^{\mathrm{Tr}} C_t z = \sum c_j z_j^2 \Longrightarrow z = 0$. Therefore $y = 0$, and $B_i C_t B_i^{\mathrm{Tr}}$ is non-singular.

Remark 2: The estimate in theorem 3 is, like the Zippel-Schwartz estimate, somewhat pessimistic. The following argument attempts to shed further light on the situation. Over a finite field of q elements we may choose the set S to be the entire field, that is, $q = \#(S)$. If we make the heuristic assumption that $a_i = f(p_1^i,\ldots,p_n^i)$ are randomly uniformly distributed, the probability that

$$0 \ne (\det(\mathcal{A}_1)\cdots\det(\mathcal{A}_t))_{\alpha_1\leftarrow a_1,\ldots,\alpha_{2t-1}\leftarrow a_{2t-1}}$$

is exactly $(1-1/q)^t \ge 1 - t/q$; cf. [12]; the proof is by induction on i, viewing $\det(\mathcal{A}_{i+1})$ as a linear polynomial in α_{2i+1} whose coefficient is $\det(\mathcal{A}_i)$. Even then, the probability of premature false termination can become unacceptably high. In our implementation, we therefore make a further modification: the user can supply a threshold $\zeta \ge 1$. Then the early termination strategy requires ζ zero discrepancies with $2L < r$ in a row. Clearly, for random a_i then there are more acceptable $\mathcal{A}_t$. The precise analysis is more complicated and carried out for Newton interpolation in theorem 4 below.

Remark 3: Our early termination methodology also applies to the sparse interpolation algorithms for Chebyshev and Pochhammer bases [14]. The theorems corresponding to theorem 3 above are similar and will be presented in the journal version of this paper.

We conclude by giving a description of the algorithm.

The Ben-Or/Tiwari algorithm with early termination

Input: f**:** *a multivariate black box polynomial.*
ζ**:** *the threshold for early termination.*

Output: c_j *and* m_j**:** *it is probabilistically correct that* $f = \sum_{j=1}^t c_j m_j$.
*An error message***:** *if fail to complete.*

1. (The early termination within the Berlekamp/Massey algorithm)

Pick random elements $\{p_1, \ldots, p_n\}$.
Execute the Berlekamp/Massey algorithm on $a_i = f(p_1^i, \ldots, p_n^i)$ *for* $i = 1, 2, \ldots$. *If* $\Delta_r = 0$ *and* $2L < r$, *that* ζ *many times in a row, then break out of the loop. Set* $\Lambda(z)$ *to the reverse of* $\Lambda_r(z)$ *that was computed inside the algorithm.*

2. (Determine m_j)
Compute all roots of $\Lambda(z)$ *in the domain of the* p_i*'s. If* $\Lambda(z)$ *does not completely factor, the early termination was false.*
From the roots, b_j, *determine the terms* m_j: *e.g., repeatedly divide* b_j *by* $p_1, \ldots, p_n$. *Again, the term recovery might fail for unlucky* p_i*'s.*

3. (Determine c_j)
Solve the transposed Vandermonde system (1) to recover the coefficients of the terms.
End.

3. EARLY TERMINATION ALGORITHMS EMBEDDED IN ZIPPEL'S ALGORITHM

3.1 Newton interpolation vs. the Ben-Or/Tiwari algorithm with early termination

We now explain the early termination of interpolations and how a higher threshold can improve its probability of success.

THEOREM 4. **(Early Termination of Univariate Interpolation)** *Given are a black box univariate polynomial* $f(x)$, δ, *an upper bound of* $\deg(f)$, *and a positive integer threshold* η. *Let* $p_0, p_1, p_2, \ldots$ *be chosen randomly and uniformly from a subset* S *of the domain of values, and let* $f^{[i]}$ *denote the interpolation polynomial that interpolates* $f(p_0), \ldots, f(p_i)$. *Note that the* p_i *are not necessarily all distinct. Suppose there is a* $d \leq \delta$ *with*

$$f^{[d]} = f^{[d+1]} = \cdots = f^{[d+\eta]}. \tag{6}$$

Then with probability at least

$$1 - (d+1)\left(\frac{\deg(f)}{\#(S)}\right)^{\eta}$$

$f^{[d]}$ *correctly interpolates* f.[‡]

Proof: Suppose d is the smallest integer that satisfies (6) and let $\mu = d + \eta$. Then $f^{[d]}$ does not correctly interpolate f if:

1. either $d = 0$, or p_d is not a root of $f - f^{[d-1]}$; and

2. $p_{d+1}, \ldots, p_{d+\eta}$ are all roots of $f - f^{[d]}$.

Since $f^{[i]}$ interpolate the values of f and $f^{[d]} \neq f$, $\deg(f^{[d]}) < \deg(f) \leq \delta$. Therefore, $\deg(f - f^{[d]}) = \deg(f)$ and there are at most $\deg(f)$ distinct roots of $f - f^{[d]}$. The probability of randomly generating an element from a set S that is a root of $f - f^{[d]}$ is no more than $\deg(f)/\#(S)$.

For $1 \leq i \leq \mu - \eta$, le $P(i)$ denote the probability that $f^{[i]} \neq f$ and i is the smallest number such that p_i is not a root of $f - f^{[i-1]}$ but all $p_{i+1}, \ldots, p_{i+\eta}$ are roots of $f - f^{[i]}$. When $i = 0$, let $P(0)$ be the probability that $f^{[0]} \neq f$ but $p_1, \ldots, p_\eta$ are all roots of $f - f^{[0]}$. It is clear that $\sum_{i=0}^{\mu-\eta} P(i)$ covers all the possibilities of f being falsely interpolated. Here we do not need to consider $P(i)$ for $i \geq \mu - \eta + 1$, since there will not be enough remaining elements in $p_{\mu-\eta+2}, \ldots, p_\mu$ following any such p_i for condition (6).

For $0 \leq i \leq \mu - \eta$, $P(i) \leq (\deg(f)/\#(S))^{\eta}$, because at least we need to continuously pick a random number being a root of $f - f^{[i]}$ η many times so that $p_{i+1}, \ldots, p_{i+\eta}$ are all roots of $f - f^{[i]} = 0$. Therefore, $f^{[d]}$ correctly interpolates f with probability at least:

$$1 - \sum_{i=0}^{\mu-\eta} P(i) \geq 1 - (\mu - \eta + 1)\left(\frac{\deg(f)}{\#(S)}\right)^{\eta}. \quad \boxtimes$$

The Newton interpolation always returns an interpolant for the given inputs and degree bound. In the above early termination case, it needs at least $1 + \deg(f) + \eta$ inputs. But when f is sparse, the number of terms might be far smaller than the degree, and the Ben-Or/Tiwari algorithm might determine the interpolating polynomial within fewer inputs, namely $2t + \zeta + 1$ where t is the number of terms in f. However, the early termination Ben-Or/Tiwari algorithm may fail at some intermediate step. Therefore, on the same black box probes, our algorithm implements both the Newton and the Ben-Or/Tiwari algorithms.

We first randomly pick an element p from a finite subset of the coefficient domain and set a_0 to be the black box probe on p, namely $a_0 = f(p)$. Let $a_i = f(p^{i+1})$ be the next black box probe added to the sequence $a_0, \ldots, a_{i-1}$. We compute both the Newton interpolant $f_N^{[i]}$ and the error locator polynomial Λ_i in the Berlekamp/Massey algorithm on $a_0, \ldots, a_i$, and check whether Λ_i or $f_N^{[i]}$ satisfies the early termination conditions. If neither does, we proceed with the next i. If $f_N^{[i]}$ satisfies the early termination conditions, we interpolate f as $f_N^{[i]}$; else if Λ_i satisfies early terminations, we proceed with all the remaining steps of the Ben-Or/Tiwari algorithm on Λ_i. If we can successfully finish all the steps, we have f interpolated with high probability. Otherwise, we again pick an new p randomly from the finite subset of the coefficient domain, and start to work Λ_j on a new sequence $\tilde{a}_0, \ldots, \tilde{a}_j$ where $\tilde{a}_k = f(p^{k+1})$. However, we still update the Newton interpolant on the sequence of all the black box probes we have acquired so far; $a_0, \ldots, a_i, \tilde{a}_0, \ldots, \tilde{a}_j$ as it is now.

Since our algorithm uses the interpolation points p^i, Theorem 4 does not directly apply. Furthermore, in this sequence of black box probes, a point might be repeated. If this happens, our Newton interpolation will not update the Newton interpolant (see below). The overall algorithm, however, will never run into an infinite loop since it is bounded in a for-loop up to $\delta + \eta$, where δ is an upper bound of $\deg(f)$ and η is the threshold in Newton interpolation.

[‡]Note added on Apr 24, 2001: If p_i are not necessarily all distinct, we actually have $1 - \eta \cdot \deg(f)\left(\frac{\deg(f)}{\#(S)}\right)^{\eta}$ instead of $1 - (d+1)\left(\frac{\deg(f)}{\#(S)}\right)^{\eta}$, which would require p_i all distinct.

The Ben-Or/Tiwari algorithm might terminate earlier, but might not finish at all; the Newton interpolation might take more black box probes than the Ben-Or/Tiwari algorithm, but can always finish interpolating. Therefore, we race these two algorithms in our univariate interpolation algorithm to take advantage of both algorithms, or to compensate for the disadvantage of either one. That is, when it is possible, our algorithm can always terminate earlier while its termination is still guaranteed. Moreover, we take advantage of the information from both algorithms at the same time. We use some information acquired from one algorithm to check the result of the other algorithm, e.g., to see whether $\deg(f)$ of f recovered from the Ben-Or/Tiwari algorithm is less than or equal to the degree of most recent Newton interpolation $f^{[i]}$, namely $\deg(f^{[i]})$.

An algorithm that races Newton against Ben-Or/ Tiwari with early termination

Input: f**:** *a univariate black box polynomial over* $\mathbb{K}$
δ**:** *an upper bound of* $\deg(f)$
η**:** *the threshold of Newton interpolation.*
ζ**:** *the threshold in deciding the error locator polynomial in Berlekamp/Massey algorithm embedded in Ben-Or/Tiwari interpolation algorithm.*

Output: $\tilde{f} = \sum_{i=0}^{d} c_i x^i$**:** *with probabilistically correctness,* $\tilde{f} = f$.

1. (Initialization.)
   $0 \neq p$ *random,* $p \in S \subseteq \mathbb{K}$*;* $a_0 \leftarrow f(p)$*;* $\tilde{a}_0 \leftarrow a_0$*;*
   $new_{[race]} \leftarrow$ *false;* $j \leftarrow 0$*;* $k \leftarrow 0$*;*
   Initialize Newton interpolant $f_N^{[0]}$ *at* a_0*,* $f_N^{(0)} \leftarrow f_N^{[0]}$
   Initialize Λ_0*,* L*,* B *at* $\tilde{a}_0$*.*

2. (Interpolate at one more point)
   For $i = 1, \ldots, \delta + \eta$ ***Do***
   If $new_{[race]} =$ *false then*
   $j \leftarrow j + 1; a_i \leftarrow f(p^{j+1}); \tilde{a}_j \leftarrow f(p^{j+1});$
   Update Newton interpolant $f_N^{[i]}$ *on* a_0*,* a_1*, ...,* a_i*.*
   If $a_i \notin \{a_0, \ldots, a_{i-1}\}$ *then*
   $k \leftarrow k + 1$*;*
   $f_N^{(k)} \leftarrow f_N^{[i]}$*;*
   Update Λ_j*,* L*,* B *on* $\tilde{a}_0$*,* $\tilde{a}_1$*, ...,* $\tilde{a}_j$*;*
   Else
   $j \leftarrow 0$*; randomly generate a nonzero* p *from* $S \subseteq \mathbb{K}$*;*
   $\text{new}_{[\text{race}]} \leftarrow$ false; $a_i \leftarrow f(p); \tilde{a}_0 \leftarrow f(p);$
   Update Newton interpolation $f_N^{[i]}$ *on* $a_0, a_1, \ldots, a_i$*.*
   If $a_i \notin \{a_0, \ldots, a_{i-1}\}$ *then*
   $k \leftarrow k + 1$*;*
   $f_N^{(k)} \leftarrow f_N^{[i]}$*;*
   Initialize Λ_0*,* L*,* B *at* $\tilde{a}_0$*;*

3. (See whether the Ben-Or/Tiwari algorithm finishes before the Newton interpolation)
   If $f_N^{(k)} = f_N^{(k-1)} = \cdots = f_N^{(k-\eta)}$ *then*
   Return $\tilde{f} \leftarrow f^{(k)}$;
   Else if $\Delta_{\tilde{j}} = 0$ *and* $2L_{\tilde{j}} \leq \tilde{j} + 1$ *and* $\tilde{j} > 1$ *for* $\tilde{j} = j - \zeta, \cdots, j$*, then*
   $\Lambda(z) \leftarrow \Lambda_j$*;*

4. (Now we attempt to complete the Ben-Or/Tiwari algorithm in subsection 2.3 on $\Lambda(z)$. If we fail to complete, we set $\text{new}_{[\text{race}]}$ as true in order to initiate a new random p and its power sequence for the "racing".)
   Complete the step 2 and 3 of the Ben-Or/Tiwari algorithm in subsection 2.3 on $\Lambda(z)$*.*
   If fail to complete, then
   $new_{[race]} \leftarrow$ *true;*

End For;
If $\tilde{f}$ *is not defined then Fail;*
End.

In step 4 of the above algorithm, we may fail to complete the Ben-Or/Tiwari algorithm if any of the following occurs: not all $\deg(\Lambda)$ many roots of $\Lambda(z)$ are powers of an element in S, or there exists a multiple root, or the transposed Vandermonde system is singular.

Notice in Newton interpolation, we compare $f_N^{(k)}, \ldots, f_N^{(k-\eta)}$ instead of $f_N^{[i]}, \ldots, f_N^{[i-\eta]}$ for early termination. Since the variable k will be increased only if the Newton interpolant $f_N^{[i]}$ interpolates at a new point, $f_N^{(k)}$ is only updated when a new point is introduced and will delay the update at a repeated point. This change avoids some unnecessary false early termination due to interpolating at repeated points, hence improves the probability of success.

3.2 The Zippel algorithm

The *sparse representation* of a black box polynomial f is as follows:

$$f(x_1, \ldots, x_n) = \sum_{(e_1, \ldots, e_n) \in J} c_{e_1, \ldots, e_n} x_1^{e_1} \cdots x_n^{e_n}, \qquad (7)$$

where $0 \neq c_{e_1,\ldots,e_n} \in \mathbb{K}$, $J \subseteq (\mathbb{Z}_{\geq 0})^n$. Here $\mathbb{Z}_{\geq 0}$ is the set of nonnegative integers. Note that $\#(J)$ is the number of monomials in f.

The Zippel algorithm is based on the following idea: during the variable by variable interpolation, a zero coefficient is the image of a zero polynomial with high probability. We present the Zippel algorithm now:

The Zippel algorithm

Input: f**:** *a multivariate black box polynomial over* $\mathbb{K}$
$(x_1, \ldots, x_n)$**:** *an ordered list of variables in* f
δ**:** *an upper bound of* $\deg(f)$

Output: $\sum_{(e_1,\ldots,e_n) \in J} c_{e_1,\ldots,e_n} x_1^{e_1} \cdots x_n^{e_n}$**:** *which equals* f *with high probability.*

1. (Initialize the anchor points.)
   Randomly pick $a_2, \ldots, a_n$ *from a finite subset* $S \subseteq \mathbb{K}$*;*

2. (Interpolating one more variable: with high probability, we have

$$f(x_1,\ldots,x_{i-1},a_i,\ldots,a_n) = \sum_{(e_1,\ldots,e_{i-1})\in J_{i-1}} c_{e_1,\ldots,e_{i-1}} x_1^{e_1}\cdots x_{i-1}^{e_{i-1}-1},$$

where $0 \neq c_{e_1,\ldots,e_{i-1}} \in \mathbb{K}, J_{i-1} \subset \mathbb{Z}_{\geq 0}^{i-1}$.)
For $i = 1,\ldots,n$ ***Do***

(Update the degree upper bound for monomials in x_i.)
$\delta_i = \max\{\delta - e_1 - \cdots - e_{i-1} \mid (e_1,\ldots,e_{i-1}) \in J_{i-1}\}$;
(Update the number of monomials in $x_1,\ldots,x_{i-1}$.)
$j_{i-1} \leftarrow \#(J_{i-1})$;

3. (Interpolate one more degree on the coefficient polynomials. We consider the coefficients of f in the variables $x_1,\ldots,x_{i-1}$ as polynomials in $\mathbb{K}[x_i]$ and interpolate those polynomials at $b_0,\ldots,b_{\delta_i}$ in the for loop in k below, because those polynomials are all of degree no more than δ_i.)
For $k = 0,\ldots,\delta_i$ ***Do***

Randomly pick b_k from a subset of $\mathbb{K}$;

4. (For every b_k, solving the following j_{i-1} by j_{i-1} transposed Vandermonde system can locate the value of every such coefficient polynomial evaluated at $x_i = b_k$.)
Set up a j_{i-1} by j_{i-1} transposed Vandermonde system:
For $j = 0,\ldots,j_{i-1}-1$ ***Do***

$$\sum_{(e_1,\ldots,e_{i-1})\in J_{i-1}} \gamma_{e_1,\ldots,e_{i-1},k}(\tilde{a}_1^j)^{e_i}\cdots(\tilde{a}_{i-1}^j)^{e_i} = f(\tilde{a}_1^j,\ldots,\tilde{a}_{i-1}^j,b_k,a_{i+1},\ldots,a_n); \quad (8)$$

End *j* ***For;***
If the system is singular then report "Failure";
Else solve the system for all $\gamma_{e_1,\ldots,e_{i-1},k}$ [11];

5. (Next we interpolate j_{i-1} many univariate polynomials in x_i. Those polynomials are the coefficients in $\mathbb{K}[x_i]$ of terms in the variables $x_1,\ldots,x_{i-1}$ and each evaluates to $\gamma_{e_1,\ldots,e_{i-1},k}$ at b_k. We excute Newton algorithm for interpolation).
For every $(e_1,\ldots,e_{i-1}) \in J_{i-1}$ ***Do***

Execute Newton interpolation so that
$c^{[k]}_{i,(e_1,\ldots,e_{i-1})}(x_i) \in \mathbb{K}[x_i]$ *and*
$c^{[k]}_{i,(e_1,\ldots,e_{i-1})}(b_s) = \gamma_{e_1,\ldots,e_{i-1},s}$, $0 \leq s \leq k$;
$c^{[k]}_{i,(e_1,\ldots,e_{i-1})}(x_i) \leftarrow \sum_{s=0}^{k} c_{i,(e_1,\ldots,e_{i-1}),s} x_i^s$;

End $(e_1,\ldots,e_{i-1})$ ***For;***

End *k* ***For;***

6. (Prune all the monomials with zero coefficient and update J_i).
$J_i = \emptyset$;
For every $(e_1,\ldots,e_{i-1}) \in J_{i-1}$ ***and*** $s = 0,\ldots,\delta_i$ ***Do***

If $c^{[\delta_i]}_{i,(e_1,\ldots,e_{i-1}),s} \neq 0$ *then*

$c_{e_1,\ldots,e_{i-1},s} \leftarrow c^{[\delta_i]}_{i,(e_1,\ldots,e_{i-1}),s}$;
$J_i \leftarrow J_i \cup \{(e_1,\ldots,e_{i-1},s)\}$;

End $(e_1,\ldots,e_{i-1}),s$ ***For;***
Randomly pick $\tilde{a}_i$ from a subset of $\mathbb{K}$;

End *i* ***For;***
End.

3.3 Homogenizing variable and pruning

In the above Zippel algorithm, the monomials with zero coefficient are *pruned*; they are dropped and assumed to be zero polynomials in all other variables. This is true with high probability if the anchor points are chosen at random. We prune each time after a new variable gets interpolated.

A new variable x_0, the *homogenizing variable*, can be introduced into the *sparse representation* (7) of f as follows:

$$\tilde{f}(x_0,x_1,\ldots,x_n) = f(x_0x_1,x_0x_2,\ldots,x_0x_n) = \sum_{(e_1,\ldots,e_n)\in J} (c_{e_1,\ldots,e_n}x_1^{e_1}\cdots x_n^{e_n})x_0^{e_1+e_2+\cdots+e_n}$$

Instead of interpolating $f(x_1,\ldots,x_n)$, we interpolate $\tilde{f}(x_0, x_1, \ldots, x_n)$. Let the anchor points $a_1,\ldots,a_n$ be randomly picked from a subset of $\mathbb{K}$. That is, we start interpolating $\tilde{f}(x_0,a_1,\ldots,a_n)$ with respect to x_0 and get a polynomial $f_0(x_0)$ in $\mathbb{K}[x_0]$. By the Zippel algorithm, in $f_0(x_0)$ we can prune the support structure of f.

Let $f_0(x_0) = \sum_{k=0}^{d} \gamma_{0,k}x_0^k$, by definition, $\gamma_{0,k}$ is the image of polynomial $c_k(x_1,\ldots,x_n) \in \mathbb{K}[x_1,\ldots,x_n]$ at the anchor point, namely $c_k(a_1,\ldots,a_n) = \gamma_{0,k}$, and

$$c_k(x_1,\ldots,x_n) = \sum_{e_1+\cdots+e_n=k,(e_1,\ldots,e_n)\in J} c_{e_1,\ldots,e_n}x_1^{e_1}\cdots x_n^{e_n}.$$

Notice that every term in c_k is of degree k in $\mathbb{K}[x_1,\ldots,x_n]$. Moreover, if f_0 is correctly interpolated, $\deg(f_0)$ in $\mathbb{K}[x_0]$ is equivalent to $\deg(f)$ in $\mathbb{K}[x_1,\ldots,x_n]$. The degree of every non-zero monomial in $f_0(x_0)$ provides an upper bound for the degree of all the intermediate terms of its coefficient polynomial.

Now, we refine the pruning idea in [4]. Comparing to Zippel's idea, we will do two more types of pruning so that we possibly reduce the size of the transposed Vandermonde system in step 4 of the above Zippel algorithm.

During the process of interpolating the homogenized polynomial in a variable by variable manner, as step 2 in the above Zippel algorithm, with high probability we have

$$\tilde{f}(x_0,x_1,\ldots,x_{i-1},a_i,\ldots,a_n) = \sum_{(e_0,e_1,\ldots,e_{i-1})\in J_{i-1}} c_{e_0,e_1,\ldots,e_{i-1}}x_1^{e_1}\cdots x_{i-1}^{e_{i-1}}x_0^{e_0},$$

where $0 \neq c_{e_0,e_1,\ldots,e_{i-1}} \in \mathbb{K}, J_{i-1} \subset (\mathbb{Z}_{\geq 0})^i$.

For every $(e_0,\ldots,e_{i-1}) \in J_{i-1}$ such that $e_1+\cdots+e_{i-1} = e_0$, the degree of the corresponding coefficient monomial has reached the total degree upper bound $e_0 = e_1 + \cdots + e_n$. That is, $c_{e_0,e_1,\ldots,e_{i-1}}x_1^{e_1}\cdots x_{i-1}^{e_{i-1}}x_0^{e_0}$ is an actual monomial in $\tilde{f}$. All such coefficient monomials have been fully interpolated and will not be changed in the further interpolations in other variables. We now let $g_{i-1}(x_0,x_1,\ldots,x_n)$ denote a polynomial summing up all such fully interpolated monomials in $x_0,\ldots,x_{i-1}$, and form the set J'_{i-1} from J_{i-1} by

removing all $(e_0, \ldots, e_{i-1})$'s such that $e_1 + \cdots + e_{i-1} = e_0$. The equation (8) in step 4 of Zippel's algorithm now becomes

$$f(\tilde{a}_0^j, \ldots, \tilde{a}_{i-1}^j, b_k, a_{i+1}, \ldots, a_n) = \sum_{(e_0,\ldots,e_{i-1}) \in J'_{i-1}} \gamma_{e_0,\ldots,e_{i-1},k} (\tilde{a}_1^j)^{e_1} \cdots (\tilde{a}_{i-1}^j)^{e_{i-1}} (\tilde{a}_0^j)^{e_0} + g_{i-1}(\tilde{a}_0^j, \ldots, \tilde{a}_{i-1}^j, b_k, a_{i+1}, \ldots, a_n)$$

Since $\#(J'_{i-1}) \leq \#(J_{i-1})$, by subtracting $g_{i-1}(\tilde{a}_0^j, \ldots, \tilde{a}_{i-1}^j, b_k, a_{i+1}, \ldots, a_n)$ from both sides of the transposed Vandermonde system, we only need to solve a smaller system. All the monomials in g_{i-1} are called *permanently pruned* since they will not be interpolated in variables $x_i, \ldots, x_n$.

When we are interpolating the coefficients of terms in $x_0, \ldots, x_{i-1}$ of $\tilde{f}(x_0, \ldots, x_{i-1}, x_i, a_{i+1}, \ldots, a_n)$, those coefficients are different polynomials in $\mathbb{K}[x_i]$. Some of those coefficient polynomials might be determined early via early termination in either Ben-Or/Tiwari or in Newton. Thus, their values can be taken out of the loop in step 3 before the rest of the coefficients (in x_i) are completed. Similarly to permanently pruned monomials, we may prune these terms *temporarily* and reduce the dimensions of the Vandermonde system (8) further.

We are now ready to present our hybrid Zippel algorithm.

3.4 Modular techniques and hybrid of Zippel's sparse interpolation algorithm

In order to control the size of the coefficients in the error locator polynomial of the embedded Berlekamp/Massey algorithm, Kaltofen *et al.* [10] apply modular techniques for finding $\Lambda(z)$ and locating the roots of $\Lambda(z)$. However, to provide a modular image of $\Lambda(z)$ sufficient for recovery of the m_i, the modulus q needs to be *sufficiently large*. They use a modulus p^k that is larger than each b_j, the m_j evaluated a prime numbers. Now consider a multivariate black box polynomial f and suppose let $d = \deg(f)$; since 2 is the smallest prime, a sufficiently large modulo p^k is at least 2^d. This means when $\deg(f)$ is relatively large, we need to perform all computations modulo an integer of length proportional to the degree, even though the coefficients could be a much small size.

However, we notice that for the recovery of the terms of a sparse univariate polynomial $\tilde{f}(x)$ (with degree $\tilde{d}$), a prime $\tilde{q}$ larger than $\tilde{d}$ can already provide a *sufficiently large* modulus. We may evaluate the single variable x at a primitive root ϱ and recover the term exponents as the discrete logarithms of the b_js. There are $\phi(\tilde{q}-1)$ primitive roots modulo $\tilde{q}$, with $u/\text{loglog}u = O(\phi(u))$ for any integer u and ϕ denoting Euler's function [9, sec. 18.4], so a random residue has a fair chance of being a primitive root.

Our algorithm picks a random residue ϱ for x and for each b_k tries the exponents $e_k = 0, 1, 2, \ldots$ until $\varrho^{e_k} \equiv b_k \pmod{q}$. The method produces an incorrect term exponent if for ϱ, ϱ^2, …, $\varrho^{\lambda_q(\varrho)} \equiv 1 \pmod{q}$ we have $e_k > \lambda_q(\varrho)$. However, such false exponent computations highly likely lead to inconsistencies in later steps. An immediate inconsistency is to the degrees of the concurrent Newton interpolation (see subsection 3.1). In that case, the algorithm even recovers the univariate intermediate result by Newton interpolation and continues. If the false exponent is not caught then, with high likelihood the inconsistency shows up later, at the latest during the comparison of the final sparse interpolant with the black box input at an additional random point.

The trade-off between the size of the modulus and the number of black box probes in Ben-Or/Tiwari versus Zippel with univariate Ben-Or/Tiwari, both with early termination, can now be quantified. Ignoring the size of the coefficients, in the former we have $q > 2^{\deg(f)}$ versus $q = O(\deg(f))$, while the number of probes is $2t + \zeta$ versus $O(n(2t + \zeta))$. Therefore, if the degrees are small but there are many variables, the pure Ben-Or/Tiwari (with early termination) may still outperform Zippel. We add that at any stage in the variable-by-variable Zippel interpolation algorithm, we could complete the rest of the variables by a multivariate Ben-Or/Tiwari algorithm, but we have not done this yet.

Our implementation of sparse interpolation with coefficients from a sufficiently large finite field uses the Zippel technique as the outer loop. Each univariate interpolation step implements a race between Newton interpolation and the Ben-Or/Tiwari algorithm (see subsection 3.1), where the Ben-Or/Tiwari algorithm recovers the term structure by the tricks explained above. Finally, we introduce a homogenizing variable and perform both permanent and temporary term pruning as discussed in subsection 3.3. We note that for small finite coefficient fields, say $\mathbb{Z}/2\mathbb{Z}$, one can switch to the coefficient domain $\mathbb{Z}/2\mathbb{Z}[x_n]$, where x_n is the last variable, and proceed modulo irreducible polynomials in $\mathbb{Z}/2\mathbb{Z}[x_n]$.

4. MAPLE IMPLEMENTATION

The *protobox* package is our Maple V.5 implementation of this new hybrid algorithm. We present part of the performance tests here. Additional data executed on Maple 6 will be presented in the journal version of this paper. Table 1 lists the polynomials used in the performance tests described below. Note that f_1 and f_2 are from [23, p. 100] and f_3 and f_4 from [23, p. 102].

4.1 Black box probes

We can "turn off" either Newton or Ben-Or/Tiwari by setting the corresponding thresholds to ∞ and force all the interpolations performed by the remaining active one. Now we interpolate each corresponding polynomial under different embedded algorithms on a relatively large modulus 100003. Table 2 lists the average number of black box probes needed after ten runs. Both Newton and Ben-Or/Tiwari thresholds are set to one by default. Our "racing" algorithm always takes advantage of both the Newton and Ben-Or/Tiwari algorithms; the average black box probes needed is never more than the minimum of either one.

4.2 Thresholds and moduli

In addition to the thresholds for Newton and Ben-Or/Tiwari, in *protobox* we introduce additional thresholds in order to further improve the probability of success and the throughput of the overall algorithm.

One way to further improve the probability of correctness of the interpolating polynomial is to pick a few more random

$f_1(x_1,\ldots,x_{10})$	$x_1^2x_3^3x_4x_6x_8x_9^2 + x_1x_2x_3x_4^2x_5^2x_8x_9 + x_2x_3x_4x_5^2x_8x_9 + x_1x_3^3x_4^2x_5^2x_6^2x_7x_8^2 + x_2x_3x_4x_5^2x_6x_7x_8^2$
$f_2(x_1,\ldots,x_{10})$	$x_1x_2^2x_4^2x_8x_9^2x_{10}^2 + x_2^2x_4x_5^2x_6x_7x_9x_{10}^2 + x_1^2x_2x_3x_5^2x_7^2x_9^2 + x_1x_3^2x_4^2x_7^2x_9^2 + x_1^2x_3x_4x_7^2x_8^2$
$f_3(x_1,\ldots,x_{10})$	$9x_2^3x_3^3x_5^2x_6^2x_8^3x_9^3 + 9x_1^3x_2^2x_3^3x_5^2x_7^2x_8^2x_9^3 + x_1^4x_3^4x_4^2x_5^4x_6^4x_7x_8^5x_9 + 10x_1^4x_2x_3^4x_4^4x_5^4x_7x_8^3x_9 + 12x_2^3x_4^3x_6^3x_7^2x_8^3$
$f_4(x_1,\ldots,x_{10})$	$9x_1^2x_3x_4x_6^3x_7^2x_8x_{10}^4 + 17x_1^3x_2x_5^2x_6^2x_7x_8^3x_9^4x_{10}^3 + 17x_2^2x_3^4x_4^2x_7^4x_8^3x_9x_{10}^3 + 3x_1^3x_2^2x_6^3x_{10}^2 + 10x_1x_3x_5^2x_6^2x_7^4x_8^4$
$f_5(x_1,\ldots,x_{20})$	$\sum_{i=1}^{20} x_i^{20}$
$f_6(x_1,\ldots,x_5)$	$\sum_{i=0}^{5}(x_1x_2x_3x_4x_5)^i$
$f_7(x_1,x_2,x_3)$	$x_1^{20} + 2x_2 + 2x_2^2 + 2x_2^3 + 2x_2^4 + 3x_3^{20}$

Table 1: Polynomials tested in Maple application program.

	mod	Newton	Ben-Or/ Tiwari	"Racing"
f_1	100003	147	137	126
f_2	100003	146	143	124
f_3	100003	209	143	133
f_4	100003	188	149	133
f_5	100003	462	101	101
f_6	100003	152	88	88
f_7	100003	95	47	42

Table 2: Average number of black box probes needed for different embedded univariate interpolation algorithms after 10 runs.

points and check whether the input black box and the interpolating polynomial agree at all those points. If they do not, an error message will be reported indicating the interpolating polynomial cannot pass the post test. The number of the random points used for the post test is defined as the optional argument "posttest_thresh" and is by default zero.

In the Zippel algorithm, we may correctly interpolate all the monomials in the previous variables yet still encounter problems. This could happen due to a bad random point that makes two different terms map to a same value, thus yielding a singular Vandermonde system (8). Trying a few more random points to see whether the Vandermonde system is still singular might avoid some unnecessary failures and hence improve the throughput. The optional argument "mapmon_thresh", set by default to zero, defines the number of random points to be tried before the algorithm reports failure.

Regarding the delay in updating the Newton interpolants at repeated points (see subsection 3.1), an optional argument "rndrep_thresh" extends the upper bound of each univariate interpolation loop and therefore lowers the failure rate due to repeated points.

Table 3 lists the runs under different thresholds and moduli with our "racing" algorithm as the embedded univariate interpolation. We run a test which executes our probabilistic algorithm 100 times on each set of arguments. The numbers under "=" column record the number of times of successful interpolations, under "≠" record the times a polynomial is wrongly interpolated, and under "!" the number of times an error message is returned. We use Greek letters denote the thresholds as follows; τ the "posttest_thresh", κ the "mapmon_thresh", and γ the "rndrep_thresh". Theorem 4 shows that a higher threshold can improve the probability of success for early termination in univariate interpolation. That improvement, of course, carries over to the overall algorithm.

If a multivariate polynomial has degrees in every single variable that are smaller than the total degree, some relatively small moduli might suffice for interpolating such a polynomial provided we "turn off" the homogenizing variable modification (see section 3.3). Table 4 records some such interpolation runs after running 100 times in each listed case.

	Thresholds			mod 11			mod 13		
	η,ζ	τ	κ,γ	=	≠	!	=	≠	!
f_1	2	2	6	28	2	70	27	0	73
f_2	2	2	6	6	1	93	20	0	80
f_3	2	2	6	8	0	92	1	0	99
f_4	2	2	6	5	0	95	1	0	99

Table 4: The throughputs on small moduli without using a homogenizing variable.

Acknowledgements: We thank James H. Davenport and two anonymous referees for their comments.

Note: many of the authors' publications cited below are accessible through links in their Internet homepages listed under the title.

5. REFERENCES

[1] Ben-Or, M., and Tiwari, P. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. Twentieth Annual ACM Symp. Theory Comput.* (New York, N.Y., 1988), ACM Press, pp. 301–309.

[2] Brown, W. S., and Traub, J. F. On Euclid's algorithm and the theory of subresultants. *J. ACM 18* (1971), 505–514.

[3] DeMillo, R. A., and Lipton, R. J. A probabilistic remark on algebraic program testing. *Information Process. Letters 7*, 4 (1978), 193–195.

[4] Díaz, A., and Kaltofen, E. FoxBox a system for manipulating symbolic objects in black box representation. In *ISSAC 98 Proc. 1998 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 1998), O. Gloor, Ed., ACM Press, pp. 30–37.

[5] Dornstetter, J. L. On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Inf. Theory* IT-*33*, 3 (1987), 428–431.

	Thresholds			mod 31			mod 37			mod 41			mod 43			mod 47			mod 53		
	η, ζ	τ	κ, γ	=	$\neq$	!	=	$\neq$	!	=	$\neq$	!	=	$\neq$	!	=	$\neq$	!	=	$\neq$	!
f_1	1	0	0	8	2	90	7	1	92	10	3	87	7	3	90	24	2	74	22	3	75
	2	1	2	29	0	71	37	0	63	44	0	56	49	0	51	70	0	30	46	0	54
	3	2	4	34	0	66	36	0	64	46	0	54	51	0	49	79	0	21	70	0	30
f_2	1	0	0	4	3	93	4	3	93	13	0	87	14	3	83	22	4	74	23	1	76
	2	1	2	23	0	77	33	0	67	37	0	63	45	1	54	68	0	32	75	0	25
	3	2	4	44	0	56	41	0	59	51	0	49	60	0	40	81	0	19	79	0	21
f_3	1	0	0	0	2	98	0	6	94	2	5	93	4	0	96	5	0	95	6	3	91
	2	1	2	1	0	99	9	0	91	18	0	82	7	1	92	33	0	69	45	0	55
	3	2	4	19	0	81	8	0	92	18	0	82	14	0	86	52	0	48	54	0	46
f_4	1	0	0	1	4	95	0	2	98	4	2	94	8	3	89	18	2	80	11	5	84
	2	1	2	2	0	98	5	0	95	19	0	81	28	0	72	51	0	49	44	0	56
	3	2	4	5	0	95	10	0	90	31	0	69	25	0	75	80	0	20	45	0	55

Table 3: The algorithm throughputs under different modulus and thresholds

[6] Gantmacher, F. R. *The theory of matrices*, vol. 1. Chelsea publishing company, 1977.

[7] Grigoriev, D. Y., Karpinski, M., and Singer, M. F. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput. 19*, 6 (1990), 1059–1063.

[8] Grigoriev, D. Y., and Lakshman Y. N. Algorithms for computing sparse shifts for multivariate polynomials. In *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC'95* (New York, N. Y., 1995), A. H. M. Levelt, Ed., ACM Press, pp. 96–103.

[9] Hardy, G. H., and Wright, E. M. *An Introduction to the Theory of Numbers*, 5 ed. Oxford Univ. Press, Oxford, 1979.

[10] Kaltofen, E., Lakshman Y. N., and Wiley, J. M. Modular rational sparse multivariate polynomial interpolation. In *ISSAC '90 Internat. Symp. Symbolic Algebraic Comput.* (1990), S. Watanabe and M. Nagata, Eds., ACM Press, pp. 135–139.

[11] Kaltofen, E., and Lakshman Yagati. Improved sparse multivariate polynomial interpolation algorithms. In *Symbolic Algebraic Comput. Internat. Symp. ISSAC '88 Proc.* (Heidelberg, Germany, 1988), P. Gianni, Ed., vol. 358 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 467–474.

[12] Kaltofen, E., and Lobo, A. On rank properties of Toeplitz matrices over finite fields. In *ISSAC 96 Proc. 1996 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 1996), Lakshman Y. N., Ed., ACM Press, pp. 241–249.

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

[14] Lakshman Y. N., and Saunders, B. D. Sparse polynomial interpolation in non-standard bases. *SIAM J. Comput. 24*, 2 (1995), 387–397.

[15] Lakshman Y. N., and Saunders, B. D. Sparse shifts for univariate polynomials. *Applic. Algebra Engin. Commun. Comput. 7*, 5 (1996), 351–364.

[16] Massey, J. L. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory* IT-*15* (1969), 122–127.

[17] Murao, H., and Fujise, T. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. In *Proc. First Internat. Symp. Parallel Symbolic Comput. PASCO '94* (Singapore, 1994), H. Hong, Ed., World Scientific Publishing Co., pp. 304–315.

[18] Schwartz, J. T. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM 27* (1980), 701–717.

[19] Wiedemann, D. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory* IT-*32* (1986), 54–62.

[20] Zilic, Z., and Radecka, K. On feasible multivariate polynomial interpolations over arbitrary fields. In *ISSAC 99 Proc. 1999 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 1999), S. Dooley, Ed., ACM Press, pp. 67–74.

[21] Zippel, R. Probabilistic algorithms for sparse polynomials. In *Proc. EUROSAM '79* (Heidelberg, Germany, 1979), vol. 72 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 216–226.

[22] Zippel, R. Interpolating polynomials from their values. *J. Symbolic Comput. 9*, 3 (1990), 375–403.

[23] Zippel, R. E. *Probabilistic algorithms for sparse polynomials*. PhD thesis, Massachusetts Inst. of Technology, Cambridge, USA, Sept. 1979.