A Deterministic Algorithm For Sparse Multivariate Polynomial Interpolation

(Extended Abstract)

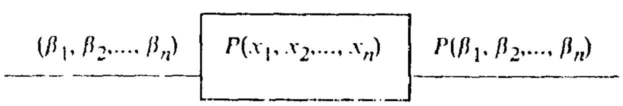
Michael Ben-Or
Hebrew University
Jerusalem, Israel

Prasoon Tiwari
I. B. M. Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract: An efficient deterministic polynomial time algorithm is developed for the sparse polynomial interpolation problem. The number of evaluations needed by this algorithm is very small. The algorithm also has a simple NC implementation.

1. Introduction

In this paper, we consider the following scenario. We are given a black-box which contains a multivariate polynomial $P(x_1, ..., x_n)$ with real (or complex) coefficients.



The black-box takes as input an n-tuple $(\beta_1, ..., \beta_n)$ and outputs the value $P(\beta_1, ..., \beta_n)$. We are also told that $P(x_1, ..., x_n)$ has at most t nonzero coefficients (i.e., it is sparse). Given this information, we must determine all the coefficients of the polynomial. This is the classical *sparse multivariate polynomial interpolation* problem.

Efficient algorithms are known for this problem which use *randomization*. Until recently, no *deterministic* polynomial time algorithm was known for this problem. A polynomial time algorithm for this problem was given by Tiwari (1987b). In this paper, we present an *efficient deterministic* polynomial time algorithm for this problem which improves the algorithm due to Tiwari (1987b). The running time of our algorithm is polynomial in the length of the output and this algorithm has an NC implementation. The main ingredients of our deterministic algorithm are: (i) a decoding algorithm for BCH codes (see, for example, Blahut, 1984), (ii) a novel technique of substituting distinct primes for various variables due to Grigoriev and Karpinski (1987), and (iii) an efficient algorithm for finding roots of polynomials which have only integer roots (Loos, 1983; Pan and Rief, 1987).

To the best of our knowledge, the best previously known algorithm for this problem is due to Zippel (1979) (see Kaltofen, 1986, for a related algorithm). The following table compares this algorithm to our new algorithm. Here t is an upper bound on the number of monomials, d is an upper bound on the degree of any variable, n is the number of variables, and ε is the probability of failure. The number of operations is counted on an algebraic RAM.

Our algorithm has a simple NC implementation. This is in contrast to the fact that Zippel's probabilistic algorithm, which performs interpolation variable by variable, is inherently sequential and requires n sequential phases for completion. Our results immediately imply the following recent results: (i) the result of Grigoriev and Karpinsky (1987) that if the number of perfect

Parameter	Zippel's Algorithm.	Our Algorithm
Type of Algorithm	Probabilistic	Deterministic
Number of Operations	ndt^3	$t^2(\log^2 t + \log nd)$
Number of Evaluations	ndt	$2t$
Size of Evaluation Points in bits	$\log(\frac{ndt}{\varepsilon})$	$t \log n$
Parallelizable	No	Yes

matchings in a graph is polynomially bounded, then they can all be determined in NC; and (ii) a generalization due to Tiwari (1987a) that if the number of full-dimensional solutions of a linear matroid parity is polynomially bounded, then they can all be determined in NC. Moreover, the total number of evaluations performed by our algorithm is linear (exactly $2t$), compared to at least quadratic evaluations in the above mentioned results, where t is the number of perfect matchings (or the full-dimensional solutions). If $P(x_1, x_2, ..., x_n)$ has integer coefficients, then we could talk of the the bit complexity of our algorithm. In this case, the bit complexity of our algorithm is polynomially bounded. Our algorithm does not require *a priori* knowledge of the degree d.

We also prove that the number of evaluations *can not* be reduced below $2t$ for a large class of interpolation algorithms, and our algorithm is in this class.

Our algorithm also sheds light on an old problem related to a problem of Edmonds (1967) (see also Lovazs 1979). Suppose A is an $n \times n$ matrix, each of whose entries is a multivariate polynomial. Is it possible to check efficiently whether $\det(A) = 0$? Even if an upper bound t on the number of monomials occurring in $\det(A)$ is known, no polynomial (in t) time algorithm is known for determining the coefficients of $\det(A)$. The obvious algorithm of evaluating the determinant may fail because some intermediate polynomials may be very dense. Our results provide a polynomial time algorithm to solve this last problem. In contrast, the problem of determining the number of terms in a multivariate polynomial which is given as the determinant of a matrix is known to be #P-Complete (Kaltofen 1986).

Another surprising result, discussed in Section 8, is as follows: Given a black-box containing a polynomial $P(\mathbf{x})$ in n variables with all coefficients *positive*, there is a *poly*(n, t, d)-time algorithm for determining $P(\mathbf{x})$ where d, and t are the (unknown) degree, and the (unknown) number of monomials appearing in $P(\mathbf{x})$. We also present a collection of interpolation problems whose complexity is open. For more recent work on related problems, also see Kaltofen and Trager (1987), and Zippel (1988).

2. Definitions and Notations

Let $P(x_1, x_2,..., x_n) = \sum_{i=1}^{t} a_i M_i(x_1,..., x_n)$, where $M_i = x_1^{\alpha_{i1}} ... x_n^{\alpha_{in}}$, are the t distinct monomials appearing in $P(x_1, x_2,..., x_n)$, and $\alpha_{ij} \in Z$, $a_i \in C$. We say that $P(x_1, x_2,..., x_n)$ is a t-sparse multivariate polynomial. Let k be the exact number of nonzero coefficients in $P(x_1, x_2,..., x_n)$. Given a black-box, the bound t such that $k \leq t$, and the number of variables n, the *sparse interpolation* problem is to determine $a_i \neq 0$ and α_{il} for $i = 1, 2, ...,k$, $l = 1,...,n$. We will also write $P(x_1, x_2,..., x_n)$ as $P(\mathbf{x})$.

We will denote the i-th prime integer by p_i. We evaluate the polynomial $P(\mathbf{x})$ at the $2t$ points given by $u_i = (p_1^i, p_2^i,..., p_n^i)$, for $i = 0,1,2,...,2t-1$. Let $v_i = P(u_i)$.

Our model of computation is an algebraic RAM. In one step, the processor can access any memory location or execute a +, −, ×, or / operation on two real numbers stored in its registers. In our algorithm, we will also need to compute a mod b where a and b are integers. We assume that this operation takes only one step. This can be done if rounding is permitted as one instruction on the RAM.

3. The Algorithm

In this section, we will consider the case $k = t$. The case $k < t$ will be resolved in the next section. Our aim is to reconstruct $P(\mathbf{x})$ using only the $2t$ numbers v_i.

Let $m_i = M_i(u_1)$, where $M_i(x_1, \ldots, x_n) = x_1^{\alpha_{i1}} \ldots x_n^{\alpha_{in}}$, is the i-th monomial appearing in $P(x_1, x_2, \ldots, x_n)$. The following important observation is due to Grigoriev and Karpinski (1986). Our algorithm relies heavily on this observation:

Observation: $m_i \neq m_j$ for $i \neq j$.

The algorithm can be partitioned into two phases. In the first phase, we determine the exponents α_{ij}, and then in the second phase we determine the coefficients a_i. In the following paragraph, we first describe a method to determine the coefficients, given the exponents. This is the easy second phase of the algorithm.

Let $\mathbf{M}$ be the $t \times t$ matrix defined by $(\mathbf{M})_{ij} = (m_j)^{i-1}$. Define $\mathbf{a}$ and $\mathbf{v}$ to be t long column vectors whose i-th components are a_i and v_{i-1} respectively. Then, the linear system $\mathbf{Ma} = \mathbf{v}$ can be solved to determine $\mathbf{a}$ because, by the above observation, $\mathbf{M}$ is a nonsingular Vandermonde matrix.

In the rest of this section, we describe the first phase of our algorithm where we find all the required exponents. This is based on a technique for decoding BCH codes (see, for example, Blahut, 1984). In order to determine the exponents involved in the i-th monomial $M_i(x_1, x_2, \ldots, x_n) = x_1^{\alpha_{i1}} x_2^{\alpha_{i2}} \ldots x_n^{\alpha_{in}}$, we determine $m_i = p_1^{\alpha_{i1}} p_2^{\alpha_{i2}} \ldots p_n^{\alpha_{in}}$ and factor it into prime powers.

In order to determine the m_i's, we define a

polynomial $\Lambda(z) = \prod_{i=1}^{t}(z - m_i) = \sum_{i=0}^{t} \lambda_i z^i$, $\lambda_t = 1$. We will show how to determine the coefficients of this polynomial by solving a linear system. Once the coefficients are known, we can determining all the m_i's by determining all the roots of $\Lambda(z)$. The linear system for determining λ_i's is derived below. Observe that:

$$
\begin{aligned}
0 &= [a_i m_i^l \Lambda(z)]_{z=m_i} \\
&= a_i\left[\lambda_0 m_i^l + \lambda_1 m_i^{l+1} + \ldots + \lambda_t m_i^{l+t}\right].
\end{aligned}
$$

Summing this over all i, we get

$$
\begin{aligned}
0 &= \sum_{i=1}^{t} a_i m_i^l \Lambda(m_i) \\
&= \lambda_0 \sum_{i=1}^{t} a_i m_i^l + \lambda_1 \sum_{i=1}^{t} a_i m_i^{l+1} + \ldots + \lambda_t \sum_{i=1}^{t} a_i m_i^{l+t} \\
&= \lambda_0 v_l + \lambda_1 v_{l+1} + \ldots + \lambda_{t-1} v_{l+t-1} + \lambda_t v_{l+t}.
\end{aligned}
$$

This last equation gives us the linear relation we want in order to determine the coefficients λ_i. Let $\mathbf{V}$ be the $t \times t$ matrix defined by $(\mathbf{V})_{ij} = v_{i+j-2}$. Define $\boldsymbol{\lambda}$ and $\mathbf{s}$ to be t-long column vectors with the i-th component given by λ_{i-1} and v_{t+i-1}, respectively. Then, by the above equations, $\mathbf{V}\,\boldsymbol{\lambda} = -\mathbf{s}$. Since $\mathbf{V}$ is a nonsingular matrix (see the next section), this system can be solved for the coefficients λ_i's.

4. The Case $k \leq t$

The analysis in the last section was restricted to the case when $k = t$, i.e., the number of nonzero coefficients in $P(\mathbf{x})$ is exactly equal to t. In this section, we extend the analysis to the case when $k < t$, i.e., t is only an upper bound on the number of nonzero coefficients in $P(\mathbf{x})$. The following lemma is the main tool in this analysis. Let $\mathbf{V}$ be the $t \times t$ matrix defined in the last section by $(\mathbf{V})_{ij} = v_{i+j-2}$. Let $\mathbf{V}_l$ be the square matrix consisting of the first l rows and columns of $\mathbf{V}$.

Theorem: If k is the exact number of monomials appearing in $P(\mathbf{x})$, then (i)

$$
\det(\mathbf{V}_l) = \sum_{S \subset \{1, 2, \ldots, k\},\ |S| = l} \{\prod_{i \in S} a_i \prod_{i > j,\ i, j \in S} (m_i - m_j)^2\},
$$

for $l \leq k$; and (ii) $\det(V_l) = 0$, for $l > k$.

Proof: Observe that $\mathbf{V}_l$ can be written as follows:

$$
\mathbf{V}_l = \begin{bmatrix} 1 & 1 & \dots & 1 \\ m_1 & m_2 & \dots & m_k \\ m_1^2 & m_2^2 & \dots & m_k^2 \\ \vdots & \vdots & & \vdots \\ m_1^{l-1} & m_2^{l-1} & \dots & m_k^{l-1} \end{bmatrix} \begin{bmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \dots & a_k \end{bmatrix} \begin{bmatrix} 1 & m_1 & m_1^2 & \dots & m_1^{l-1} \\ 1 & m_2 & m_2^2 & \dots & m_2^{l-1} \\ 1 & m_3 & m_3^2 & \dots & m_3^{l-1} \\ \vdots & & & & \vdots \\ 1 & m_k & m_k^2 & \dots & m_k^{l-1} \end{bmatrix}
$$

Clearly, $\det(V_l)$ is a polynomial in $a_1, a_2, \dots, a_k$. Let us denote this polynomial by $Q(a_1, a_2, \dots, a_k)$. We will prove part (i) of the theorem by determining the coefficients of various monomials in $Q(a_1, a_2, \dots, a_k)$:

(a) Let us determine $Q(a_1, a_2, \dots, a_r, 0, 0, \dots, 0)$ where $r < l$. Observe that if only the first $r < l$ a_i's are nonzero, then $\text{rank}(\mathbf{V}_l) = r < l$, and therefore $Q(a_1, \dots, a_r, 0, 0, \dots, 0) = 0$. Hence, $Q(\mathbf{a})$ does not contain any monomial with less than l variables.

(b) By a straight forward determinant evaluation, the total degree of each monomial in $Q(\mathbf{a})$ is exactly l. This fact, together with (a) above implies that any monomial occurring in $Q(\mathbf{a})$ is of the form $\prod_{i \in S} a_i$, where $S \subset \{1, 2, \dots, k\}$, and $|S| = l$.

(c) In order to evaluate the coefficient of $\prod_{i \in S} a_i$, set $a_i = 1$, for $i \in S$, and $a_i = 0$ otherwise. Then we see that this coefficient is infact the square of the determinant of a Vandermonde matrix.

The above argument also implies part (ii) of the theorem. ∎

Corollary: If the number of nonzero coefficients in $P(\mathbf{x})$ is bounded by t, then the number of nonzero coefficients in $P(\mathbf{x})$ equals $\max_{\mathbf{V}_j \text{ is nonsingular}, j \le t} \{j\}$.

The complete algorithm is given in Figure 1 below. Its correctness follows form the above corollary. In order to complete the description of our algorithm, we include an algorithm for finding integer roots.

5. An Algorithm for Finding Integer Roots

In Figure 2, we present an algorithm (Loos, 1983) for finding integer roots of polynomials with integer coefficients. We will need the complexity of this algorithm in order to estimate the complexity of the algorithm given in Figure 1.

Sparse Polynomial Interpolation Algorithm

Input: A black-box containing a t-sparse polynomial in n variables.
Output: All the monomials appearing in $P(\mathbf{x})$, and their coefficients.
Algorithm:
Step 1: Evaluate the polynomial at points $u_i = (2^i, 3^i, \dots, p_n^i)$, for $i = 0, 1, 2, \dots, 2t-1$. Let v_i be its value at u_i.
Step 2: Let k be the rank of the $t \times t$ matrix $\mathbf{V}$ defined by $(\mathbf{V})_{ij} = v_{i+j-2}$.
Step 3: Solve $\overline{\mathbf{V}}\, \boldsymbol{\lambda} = \mathbf{s}$, where $(\overline{\mathbf{V}})_{ij} = v_{j-i}$, $(\boldsymbol{\lambda})_i = \lambda_{i-1}$, and $(\mathbf{s})_i = v_{i+k-1}$.

Step 4: Determine the roots $m_1, \dots, m_k$ of the polynomial $\Lambda(z) = x^k + \sum_{i=0}^{k-1} \lambda_i z^i$.
Step 5: Factor $m_i = 2^{\alpha_{i1}} 3^{\alpha_{i2}} \dots p_n^{\alpha_{in}}$ to determine the monomials present the given polynomial.
Step 6: Solve $\mathbf{Ma} = \mathbf{v}$ to determine the coefficients of the given polynomial, where $(\mathbf{M})_{ij} = m_j^{i-1}$, $(\mathbf{a})_i = a_i$, and $(\mathbf{v})_i = v_{i-1}$.

Step 7: Output the polynomial $\sum_{i=1}^{k} a_i x_1^{\alpha_{i1}} x_2^{\alpha_{i2}} \dots x_n^{\alpha_{in}}$.

Figure 1.

Integer Root Finding Algorithm

Input: A monic polynomial $a(z) \in Z[z]$, $\deg a(z) = t$, whose roots are bounded by B in absolute value.
Output: The set of all integer roots of $a(z)$.
Algorithm:
Step 1: Evaluate the discriminant Δ of the given polynomial $a(z)$.
Step 2: Find the smallest prime p such that p does not divide Δ.
Step 3: By exhaustive search, find all roots of $a(z) \bmod p$. Let S be the set of roots of $a(z) \bmod p$.
Step 4: Let $S_0 = S$, and compute the set

$S_{i+1} = \{(\alpha + p^{2^i} b_\alpha) \bmod p^{2^{i+1}} \mid \alpha \in S_i,\ u = a(\alpha)/p^{2^i},\ v = (a'(\alpha))^{-1} \bmod p^{2^i},\ b_\alpha = -uv \bmod p^{2^i}\}$, where $a'(z)$ is the derivative of $a(z)$.
Step 5: Find the smallest i such that p^{2^i} is larger than B, and output the set of all integer roots of $a(z)$ from the set S_i.

Figure 2.

Let us briefly discuss the complexity of the algorithm presented in Figure 2. If $a(z)$ has degree t, then

$\Delta \leq B^{O(t^2)}$, and it can be computed in $O(t^3)$ steps. Since the product of primes less than l is at least $e^{\Omega(l)}$, we can find a prime p which is at most $O(t^2 \log B)$. Observe that $|S_i| \leq t$. Computing S_0 takes $O(pt)$ steps. Computing S_{i+1} from S_i takes $O(|S_i|(t + \log p + i))$ steps. Therefore, the number of steps taken by this algorithm on an algebraic RAM is $O(t^3 \log B)$.

6. The Number of Steps Taken by the Algorithm on the Algebraic RAM

First we analyze the algorithm as presented in Figure 1, and then we indicate how to improve the complexity. The rank in Step 2 can be determined in $O(t^3)$ steps. The resulting system can be solved in Step 3 in $O(t^3)$ steps. In Step 4, each root of $\Lambda(z)$ is at most $2^{O(dn \log n)}$, therefore $B \leq 2^{O(dn \log n)}$. As a consequence, all roots of $\Lambda(z)$, can be found in $O(t^3 dn \log n)$. The linear system in Step 6 can be solved in $O(t^3)$ steps. Therefore, the whole algorithm takes no more than $O(t^3 dn \log n)$ steps.

In the above analysis, we have used the straight forward algorithms for determining the rank, and solving the linear systems. Steps 2, 3, 4, and 6 determine the complexity of the above algorithm. The complexity of these steps can be reduced by using specialized algorithms. Steps 2 and 3 can be performed in $O(t^2)$ arithmetic operations by using Berlekamp-Massey algorithm (Blahut, 1984). Root finding in Step 4 can be accomplished in $O(t^2(\log^2 t + \log nd))$ arithmetic steps using the algorithm of Pan and Rief (1987). If implemented in this way, our algorithm for sparse multivariate polynomial interpolation takes no more that $O(t^2(\log^2 t + \log nd))$ steps on an algebraic RAM.

7. A Tight Lower Bound on the Number of Evaluations

A *nonadaptive* interpolation algorithm is an interpolation algorithm which selects the points of evaluations depending only on the given bound, t, on the number of monomials. In contrast, an *adaptive* algorithm may evaluate the polynomial at some points and then choose the next evaluation point depending upon the values attained by the polynomial at previous points. In this section, we prove that any *nonadaptive* algorithm for sparse polynomial interpolation must perform $2t$ evaluations in the worst case. Observe that our algorithm, which is nonadaptive, matches this lower bound.

In fact, this bound holds even for univariate polynomials.

Theorem : Any nonadaptive polynomial interpolation algorithm which determines a t-sparse polynomial in n variables must perform at least $2t$ evaluations.

Proof : Consider the univariate case. Suppose the interpolation algorithm evaluate the given t-sparse polynomial at $l < 2t$ points $u_1, u_2, \ldots, u_l$. Construct the polynomial $p(x) = \prod_{i=1}^{l} (x - u_i)$. Observe that $p(x) = \sum_{i=0}^{l} a_i x^i$ has almost $l + 1$ nonzero coefficients. Define $p_1(x) = \sum_{0}^{\lfloor l/2 \rfloor} a_i x^i$, and $p_2(x) = -\sum_{\lfloor l/2 \rfloor + 1}^{l} a_i x^i$. By definition, $p(x) = p_1(x) - p_2(x)$, and $p_i(x)$ is t-sparse. However, $p_1(u_i) = p_2(u_i)$, for $i = 1,2,\ldots,l$. ∎

In the light of this theorem, our algorithm is the best possible, as far as the number of evaluations are concerned.

8. The Case When No Upper Bound on The Number of Monomials is Known

In case t is not known, we could try $t = 1, 2, 3, \ldots$ but we would not know when to stop. Of course, (if the degree of $P(\mathbf{x})$ is known, then) we could use Schwartz's (1980) test to check *probabilistically* if the polynomial obtained for a particular value of t is in fact equal to $P(\mathbf{x})$. Is there some way of making this test *deterministic*?

Can we use the theorem of Section 4 in case no bound t on the number of nonzero monomials is given? It is known that the problem of determining the exact number of monomials in $P(\mathbf{x})$ given by a black-box is #P-Complete (Kaltofen 1986). However, it does not preclude the possibility of a deterministic algorithm which determines the number of monomials t in $P(\mathbf{x})$ in time $t^{O(1)}$. Indeed, the permanent of a 0/1 matrix can be evaluated in time polynomial in its value (Gal and Breitbart, 1974).

It turns out that one *can not* determine the number of monomials t in $P(\mathbf{x})$ in time $t^{O(1)}$ (this point will be discussed at length in the full paper). In light of this fact, we have the following, somewhat surprising, result:

Lemma: Given a black-box containing a polynomial $P(\mathbf{x})$ in n variables with all coefficients *positive*, there is a $poly(n, t, d)$-time algorithm for determining $P(\mathbf{x})$ where d, and t are the (unknown) degree, and the (unknown) number of monomials appearing in $P(\mathbf{x})$.

Proof: Try the algorithm of Figure 1 for $t = 1, 2, 3, \ldots$. The theorem of Section 4 provides the desired stopping rule. If $\det(V_l) > 0$, but $\det(V_{l+1}) = 0$, then $t = l$. ∎

The following lemma may give some useful information in the general case, but it falls short of providing a stopping rule:

Lemma: If the rank of $\mathbf{V}_l$ is k, then the number of nonzero coefficients in $P(\mathbf{x})$ is either exactly k, or it is at least $2l - k$.

Proof: Follows from the fact that the eigenvalues of a symmetric matrix interleave the eigenvalues of any principal minor. ∎

9. Discussion and Related Open Problems

The following lemma implies that in order to check if a univariate t-sparse polynomial is identically zero, it is sufficient to evaluate it at any points $u_i > 0$, for $i = 1,2,3,\ldots,t$.

Lemma (see, for example, Evans and Isaacs, 1976): Let $\mathbf{A}$ be a $k \times k$ matrix given by $(\mathbf{A})_{ij} = x_i^{r_j}$, where $x_i > 0$, and r_i are positive integers, for $i = 1, 2, \ldots, k$. Then, $\mathbf{A}$ is nonsingular.

It follows that in order to check if two univariate t-sparse polynomials are identical, it is sufficient to evaluate them at any $2t$ points $u_i > 0$, for $i = 1,2,\ldots,2t$. In other words, the values at these $2t$ points uniquely determines a univariate t-sparse polynomial. Picking the specific values of u_i, as we have done here, enables us to reconstruct the polynomial from these points. Can one efficiently reconstruct a univariate t-sparse polynomial from its values at any set of $2t$ points?

Rational functions give rise to another interesting interpolation problem. Define a t-sparse rational function to be $r(x) = \frac{a(x)}{b(x)}$, where $a(x) = \sum_{i=1}^{t} a_i x^{j_i}$, and $b(x) = \sum_{i=1}^{t} b_i x^{l_i}$. Then, in order to check if a t-sparse rational function is zero, it is sufficient to evaluate it at t points, $x = 1, 2, \ldots, t$, and hence check if the numerator is zero identically. Now consider the problem of checking if two t-sparse rational functions $q(x)$ and $r(x)$ are equal. By using the above lemma, we can conclude that it is sufficient to evaluate these rational functions at $2t^2$ points $x = 1, 2, \ldots, 2t^2$. Can this bound be improved? What is a good lower bound?

The problem of determining (interpolating) a t-sparse rational function, given a black-box for evaluating it, is a problem that we have not been able to solve satisfactorily. Suppose we are given a black-box for evaluating $r(x) = \frac{a(x)}{b(x)}$, where $a(x)$, and $b(x)$ are as defined above. Furthermore, assume that j_i, $l_i \leq d$ and d is given. Suppose it is known that the rational function $r(x)$ takes on values r_i at points x_i, for $i = 1, 2, \ldots, k$. Then, one way of solving the interpolation problem would be to solve the following system for a sparse vector:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d & r_1 & r_1x_1 & r_1x_1^2 & \cdots & r_1x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d & r_2 & r_2x_2 & r_2x_2^2 & \cdots & r_2x_2^d \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot \\ 1 & x_k & x_k^2 & \cdots & x_k^d & r_k & r_kx_k & r_kx_k^2 & \cdots & r_kx_k^d \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \cdots \\ g_d \\ h_0 \\ h_1 \\ \cdots \\ h_d \end{bmatrix} = \mathbf{0}.$$

Any solution of this system with at most $k/(2t)$ nonzero components will give enable us to compute $r(x)$ efficiently. The general problem of determining if there is a sparse vector in the null space of a given matrix, is known to be NP-Complete. Is there an efficient algorithm for this special case?

Generalizing our results above for the case of finite fields we encounter two major problems. The first problem is finding a replacement for the evaluation points so that later on we can recover the actual monomial from its value at these points. The second problems arises from the need to solve polynomial equations over the finite field. Here there are efficient probabilistic algorithms but we do not know efficient deterministic algorithms when the characteristic of the field is large.

Let $GF(q)$, $q = p^k$, p prime, be the finite field with q elements, and let $P(\mathbf{x}) \in GF(q)[\mathbf{x}]$ be a polynomial of degree at most d having at most t monomials. First two of the following three cases can be dealt with by our technique using only $2t$ evaluation points:
Very Large p: If the characteristic of the field is very big, say, $p > 3^{2ndt}$, we can use the algorithm for the zero characteristic case, using the same evaluation points as before. Since p is so large, we arrive at step 4 of our algorithm, with the polynomial $\Lambda(z)$, whose coefficients are integers much smaller than p. Therefore, computing modulo p, we get the same polynomial as before. This follows immediately from the fact that the product of the first n primes is less than 3^n.

We can now avoid the problem of finding the roots of polynomials over finite fields where only efficient *probabilistic* algorithms are known, by finding the roots of our polynomial $\Lambda(z)$ over the real numbers just as we did in the zero characteristic case.
Very Small p: If the characteristic p is small, say, polynomial in ndt, then we can find roots of polynomials in $GF(q^k)$ deterministically using Berlekamp's algorithm (Berlekamp, 1970) that reduces the problem to solving polynomials over the the prime field $GF(p)$, where a straight forward search can be used. The old evaluation points are useless in this case and we replace them by picking evaluation points from extension fields.

Using the algorithm due to Adelman and Lenstra (1986), we deterministically find an irreducible polynomial $f(w) \in GF(q)[w]$ of degree at least $e = 2nd$. We will use evaluation points in

$GF(q)[w]/(f(w)) \simeq GF(q^e) \simeq GF(p^{ke})$. Note that if our polynomial $P(\mathbf{x})$ is given by a polynomially long straight line program, instead of a black-box, then this computation requires only polynomially many field operations in $GF(q)$, and can also be done fast in parallel.

Without loss of generality we may assume that $q > n$, (otherwise first extend the ground field by an extension of degree $\log n$). Let $a_1, \dots, a_n$ be n distinct points in $GF(q)$. As our evaluation points we pick $u_i = ((w - a_1)^i, (w - a_2)^i, \dots, (w - a_n)^i) \in GF(q^e)^n$ for $i = 0, 1, \dots, 2t - 1$.

Let $M_i = x_1^{\alpha_{i1}} \dots x_n^{\alpha_{in}}$ be the i-th monomial of $P(\mathbf{x})$, and let $m_i = M_i(u_1)$. Since the degree of w is greater than dn, the representation of m_i as a polynomial in w modulo $f(w)$, is exactly the polynomial $m_i = (w - a_1)^{\alpha_{i1}} \dots (w - a_n)^{\alpha_{in}}$ and so $m_i \neq m_j$ for $i \neq j$. Arriving at step 4 of our algorithm we can deterministically find the roots m_i of the polynomial $\Lambda(z)$, in their representation as polynomials in w. Factoring each m_i as a polynomial in the variable w into its linear factors we can recover the actual monomial.

Intermediate p: Here we can use the same evaluation points in a large enough extension field, but we do not know how to find the roots of the polynomial $\Lambda(z)$ in an efficient deterministic way. However, using the algorithm of Tiwari (1987b), we can use the similar evaluation points in an appropriate extension field, to give a polynomial time, but less efficient, deterministic solution. This algorithm also leads to NC solution of this problem in all the above cases. However, the minimum number of evaluations required for interpolation over finite fields remains open.

10. Acknowledgements

We would like to thank Noga Alon, Allan Borodin, Don Coppersmith, Scot Hornick, Erich Kaltofen, Lakshman Yagati for discussions on topics covered in this paper. Don Coppersmith pointed out the similarity between univariate interpolation and decoding BCH codes. Discussions with Allan Borodin led to some of the open problems discussed in the paper. Noga Alon, Don Coppersmith, and Scot Hornick pointed out that, in principle, an adaptive algorithm can determine a t-sparse polynomial with only $t + 1$ evaluations. Erich Kaltofen and Lakshman Yagati pointed out that the complexity of our interpolation algorithm is dominated by the complexity of the root finding algorithm.

The second author would also like to thank Richard Karp for pointing out the work of Grigoriev and Karpinski (1986).

11. References

L. M. Adelman and H. W. Lenstra, Finding irreducible polynomials over finite fields, Proc. of the 18th Annual ACM Symposium on Theory of Computing, pp. 350-355, 1986.

E. R. Berlekamp, Factoring polynomials over large finite fields, Math. Comp. 24 (1970) 713-735.

R. E. Blahut, *The Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Co., 1983.

J. Edmonds, Systems of distinct representatives and linear algebra, J. Res. Nat. Bur. Stand. 713 (1967) 241-245.

R. J. Evans and I. M. Isaacs, Generalized Vandermonde determinants and roots of unity of prime order, Proc. of the AMS 58 (1976) 51-54.

Shmuel Gal and Yuri Breitbart, A method for obtaining all the solutions of a perfect matching problem, Technical Report No. 016, IBM Israel Scientific Center (May 1974).

J. von zur Gathen, Parallel powering, Proc. of the 25th IEEE Symposium on Foundations of Computer Science, pp. 31-36, 1984.

D. Yu. Grigoriev and M. Karpinski, The matching problem for bipartite graphs with polynomially bounded permanents is in NC, Research Report No. 857-CS, Univ. Bonn (Dec. 1986).

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

E. Kaltofen, Factorization of polynomials given by straight line programs, manuscript, 1986.

E. Kaltofen and B. Trager, Sparse factorization and rational function interpolation of polynomials given by black-boxes for their evaluation, manuscript, 1987.

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

L. Lovasz, On determinants, matchings, and random algorithms, *Fundamentals of Computing Theory,* edited by L. Budach, Akademia-Verlag, Berlin (1979).

J. T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, JACM 27 (1980) 701-717.

P. Tiwari, Parallel algorithms for instances of linear matroid parity with a small number of solutions, RC 12766, IBM T. J. Watson Research Center, May 1987.

P. Tiwari, Algorithms for multivariate polynomial interpolation, manuscript, June 1987.

R. E. Zippel, Probabilistic algorithms for sparse polynomials, Proc. EUROSAM '79, Springer Lecture Notes in Computer Science, vol. 72, pp. 216-226, 1979.

R. E. Zippel, Interpolating polynomials from their values, manuscript, 1988.