On Sparse Polynomial Interpolation over Finite Fields *

Seyed Mohammad Mahdi Javadi
School of Computing Science
Simon Fraser University
Burnaby, B.C. Canada.
sjavadi@cecm.sfu.ca.

Michael Monagan
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
mmonagan@cecm.sfu.ca.

ABSTRACT

We present a Las Vegas algorithm for interpolating a sparse multivariate polynomial over a finite field, represented with a black box. Our algorithm modifies the algorithm of Ben-Or and Tiwari in 1988 for interpolating polynomials over rings with characteristic zero to characteristic p by doing additional probes.

One of the best algorithms for sparse polynomial interpolation over a finite field is Zippel's algorithm from 1990. To interpolate a polynomial in n variables with t non-zero terms, Zippel's algorithm does $O(ndt)$ probes to the black box where d bounds the degree of the polynomial in each variable. Our new algorithm does $O(nt)$ probes.

We have implemented both Zippel's algorithm and the new algorithm in C. We provide benchmarks demonstrating the efficiency of our algorithm. We also analyze the failure probability for our algorithm.

1. INTRODUCTION

Let p be a prime and let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be a multivariate polynomial with $t > 0$ non-zero terms which is represented with a *black box* $\mathbb{Z}_p^n \to \mathbb{Z}_p$. On input $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_p^n$, the black box evaluates and outputs $f(x_1 = \alpha_1, \ldots, x_n = \alpha_n)$. Given also a degree bound d on the degree of f in each variable, our goal is to interpolate the polynomial f with minimum number of evaluations (probes to the black box).

Sparse interpolation is a key part of many algorithms in computer algebra such as GCD computation [15, 5, 2]. We are interested in algorithms whose computational complexity is polynomial in t, n, and d. The first efficient sparse interpolation algorithm is due to Richard Zippel in 1979 [15]. Zippel's algorithm is probabilistic. It relies heavily on the assumption that if a polynomial is zero at a *random* evaluation point, then it is the zero polynomial with high probability. Zippel's algorithm requires a bound on the degree of f in each variable. In his algorithm, the interpolation is done variable by variable. Zippel's algorithm makes $O(ndt)$ probes to the black box.

In 1990, Zippel in [16] improved his 1979 algorithm to use evaluation points of the form $(\alpha_1^i, \ldots, \alpha_k^i) \in \mathbb{Z}_p^k$ so that the linear systems to be solved become transposed Vandermonde systems which can be solved in $O(t^2)$ time instead of $O(t^3)$ – see [6].

In 1989, Ben-Or and Tiwari [1] presented a deterministic algorithm for interpolating a multivariate polynomial with integer, real or complex coefficients. Given a bound T on the number of terms t of the polynomial f, the algorithm evaluates the black box at powers of the first n primes; it evaluates at the points $(2^i, 3^i, 5^i, \ldots, p_n^i)$ for $0 \le i < 2T$. If $M_j(x_1, \ldots, x_n)$ are the monomials of the t non-zero terms of f, it then uses Berlekamp/Massey algorithm [12] from coding theory to find the evaluations $M_j(2, 3, 5, \ldots, p_n)$ of the monomials in f for $1 \le j \le t$ and then determines the degree of a monomial M_j in x_k by trial division of $M_j(2, 3, 5, \ldots, p_n)$ by p_j. This algorithm is not variable by variable. Instead, it interpolates the polynomial f with only $2T$ probes to the black box. The major disadvantage of the Ben-Or/Tiwari algorithm is that the evaluation points are large ($O(T \log n)$ bits long – see [1]) which makes computations slow.

In 2009, Giesbrecht, Labahn and Lee in [11] present two new algorithms for sparse interpolation for polynomials with floating point coefficients. The first is a modification of the Ben-Or/Tiwari algorithm. To avoid numerical problems, it evaluates at powers of complex roots of unity. In principle, their algorithm can be made to work over finite fields when we can choose p. The recovery of the degrees of variables requires computing a discrete logarithm where the size of the field needs to be $> d^n$. If f is modestly large, say $n = 10$ variables of degree $d < 30$, then p would need to be larger than $31 \cdot 37 \cdot 41 \cdot \cdots \cdot 71$, a 56 bit integer. The t discrete logarithms in such a field would be too expensive.

Our approach in this paper for sparse interpolation over a finite field is to use evaluation points of the form $(\alpha_1^i, ..., \alpha_n^i) \in \mathbb{Z}_p^n$ and modify the Ben-Or/Tiwari algorithm to do extra probes to determine the degrees of the variables in each monomial in f. One of the main motivations for the new algorithm is to be able to use the Ben-Or/Tiwari approach in modular algorithms (e.g. GCD computations in characteristic 0 – see [5]) where the prime p used can be selected. In the new algorithm we do a factor of at most $2n$ more evaluations in order to recover the monomials from their images.

*Supported by NSERC of Canada and the MITACS NCE of Canada

ISSAC'10, July 25–28, 2010, Munich, Germany.

We cite also the following related work. To reduce the number of probes needed, early termination versions of both Zippel's algorithm and Ben-Or/Tiwari algorithm were developed by Kaltofen *et al.* in [8, 7]. Also, the Ben-Or/Tiwari algorithm has been extended to sparsity with respect to non-standard polynomial bases (e.g. Chebyshev bases and shifted bases) [4, 9, 10, 3].

Our paper is organized as follows. In Section 2 we present an example showing the main flow and the key features of our algorithm. We then identify possible problems that can occur and how the new algorithm deals with them in Section 3. In Section 4 we present our new algorithm and analyze its time complexity. Finally, in Section 5 we compare the C implementations of our algorithm with Zippel's algorithm on various sets of polynomials.

2. THE IDEA AND AN EXAMPLE

Let $f = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be the polynomial represented with the black box with $a_i \in \mathbb{Z}_p \backslash \{0\}$. Here t is the number of non-zero terms in f. $M_i = x_1^{e_{i1}} \times x_2^{e_{i2}} \times \cdots \times x_n^{e_{in}}$ is the i'th monomial in f where $M_i \neq M_j$ for $i \neq j$. Let d be a bound on the degree of f in each variable x_i, i.e. $e_{ij} \leq d$ for all $1 \leq i, j \leq n$.

We demonstrate our algorithm on the following example. Here we use x, y and z for variables instead of x_1, x_2 and x_3.

Example 1 *Let $f = 91yz^2 + 94x^2yz + 61x^2y^2z + 42z^5 + 1$ and $p = 101$. Given the number of terms $t = 5$, the number of variables $n = 3$, a degree bound $d = 5$ and the black box that computes f, we want to find f.*

The first step is to pick $n = 3$ distinct generators $\alpha_1, \alpha_2, \alpha_3$ of $\mathbb{Z}_p^*$. We evaluate the black box at the points $\beta_0, \ldots, \beta_{2t-1}$ where $\beta_i = (\alpha_1^i, \alpha_2^i, \ldots, \alpha_n^i)$. Thus we make $2t$ probes to the black box. The reason to use generators instead of random values from $\mathbb{Z}_p$ is that it decreases the probability of two distinct monomials having the same evaluation. For our example, let the generators be

$$\alpha_1 = 66, \alpha_2 = 12 \quad \text{and} \quad \alpha_3 = 3$$

and let v_i be the output of the black box on input β_i and let $V = (v_0, \ldots, v_{2t-1})$. In this example we obtain

$$V = (87, 78, 65, 41, 49, 38, 87, 29, 23, 86).$$

Now we use the Berlekamp/Massey algorithm [12] (See [8] for a more accessible reference). The input to this algorithm is a sequence of elements $b_0, b_1, \ldots, b_{2t-1}, \ldots$ where $b_i \in \mathbb{Z}_p$. The algorithm computes a linear generator *for the sequence, i.e. the univariate polynomial $\Lambda(z) = z^t - \lambda_{t-1}z^{t-1} - \cdots - \lambda_0$ such that*

$$b_{t+i} = \lambda_{t-1}b_{t+i-1} + \lambda_{t-2}b_{t+i-2} + \cdots + \lambda_0 b_i$$

for all $i \geq 0$. In our example the input is $V = (v_0, \ldots, v_{2t-1})$ and the output is

$$\Lambda_1(z) = z^5 + 28z^4 + 62z^3 + 54z^2 + 11z + 46.$$

The next step is to find the roots of $\Lambda_1(z)$. We know (see [1]) that this polynomial is the product of exactly $t = 5$ linear factors. The roots are $r_1 = 1, r_2 = 7, r_3 = 41, r_4 = 61$ and $r_5 = 64$. Ben-Or and Tiwari prove that for each $1 \leq i \leq t$, there exists $1 \leq j \leq t$ such that

$$m_i = M_i(\alpha_1, \ldots, \alpha_n) \equiv r_j \bmod p.$$

Our goal now is to determine the degrees of each monomial in f in each variable. We do this one variable at a time starting with the first variable x. Let α_{n+1} be a new random generator of $\mathbb{Z}_p^*$. In this example we choose $\alpha_4 = 34$. This time we choose the evaluation points $\beta'_0, \ldots, \beta'_{2t-1}$ where $\beta'_i = (\alpha_{n+1}^i, \alpha_2^i, \ldots, \alpha_n^i)$. Note that this time we are evaluating the first variable at powers of α_{n+1} instead of α_1. We evaluate the black box at these points and apply the Berlekamp/Massey algorithm on the sequence of the outputs to compute the linear generator for the new sequence

$$\Lambda_2 = z^5 + 45z^3 + 54z^2 + 60z + 42.$$

Let $\bar{r}_1, \ldots, \bar{r}_5$ be distinct roots of Λ_2.
We know that $M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)$ is a root of Λ_2 for $1 \leq i \leq n$. On the other hand we have

$$\frac{M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)}{M_i(\alpha_1, \alpha_2, \ldots, \alpha_n)} = \left(\frac{\alpha_{n+1}}{\alpha_1}\right)^{e_{i1}}. \quad (1)$$

Let $r_j = M_i(\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\bar{r}_k = M_i(\alpha_{n+1}, \alpha_2, \ldots, \alpha_n)$. From Equation 1 we have

$$\bar{r}_k = r_j \times \left(\frac{\alpha_{n+1}}{\alpha_1}\right)^{e_{i1}},$$

i.e. for every root r_j of Λ_1, $r_j \times (\frac{\alpha_{n+1}}{\alpha_1})^{e_{i1}}$ is a root of Λ_2 for some e_{i1} which is the degree of some monomial in f with respect to x. This gives us a way to compute the degree of each monomial M_i in the variable x. In this example we have $\frac{\alpha_{n+1}}{\alpha_1} = 25$. We start with the first root of Λ_1 and check if $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^i$ is a root of Λ_2 for $0 \leq i \leq d$. For $r_1 = 1$ we have that $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^0$ is a root of Λ_2 and for $0 < i \leq d$, $r_1 \times (\frac{\alpha_{n+1}}{\alpha_1})^i$ is not a root of Λ_2, hence we conclude that the degree of the first monomial of f in x is 0. We continue this to find the degrees of all the monomials in f in the variable x. We obtain

$$e_1 = \{e_{11} = 0, e_{21} = 0, e_{31} = 0, e_{41} = 2, e_{51} = 2\}.$$

Now we proceed to the next variable y. This time the evaluation points used for probing the black box are $\beta''_0, \ldots, \beta''_{2t-1}$ where $\beta''_i = (\alpha_1^i, \alpha_{n+1}^i, \alpha_3^i, \ldots)$. Note that this time we are evaluating the second variable y at powers of α_{n+1} instead of α_2. We evaluate the black box at these points and apply the Berlekamp/Massey algorithm on the sequence of the outputs to compute the linear generator for the new sequence

$$\Lambda_3 = z^5 + 5z^4 + 27z^3 + 36z^2 + 93z + 40.$$

Let $\tilde{r}_1, \ldots, \tilde{r}_5$ be distinct roots of Λ_3. Again using the same approach as above, we find the degrees of the monomials in the second variable y

$$e_2 = \{e_{12} = 0, e_{22} = 1, e_{32} = 0, e_{42} = 2, e_{52} = 1\}.$$

And finally we proceed to the last variable z. This time we evaluate z at powers of α_{n+1} instead of α_3 and compute the following linear generator for the sequence of outputs obtained by probing the black box

$$\Lambda_4 = z^5 + 27z^4 + 99z^3 + 18z^2 + 16z + 41.$$

We compute the degrees with the same technique

$$e_3 = \{e_{13} = 0, e_{23} = 2, e_{33} = 5, e_{43} = 1, e_{53} = 1\}.$$

At this point we have computed all the monomials. Recall that $M_i = x_1^{e_{i1}} \times x_2^{e_{i2}} \times \cdots \times x_n^{e_{in}}$ hence we have

$$M_1 = 1, M_2 = yz^2, M_3 = z^5, M_4 = x^2y^2z \quad \text{and} \quad M_5 = x^2yz.$$

Now we need to compute the coefficients. We can easily do this by solving one linear system of equations. We computed the roots of Λ_1 and we have computed the monomials such that $M_i(\alpha_1,\ldots,\alpha_n)=r_i$. Recall that v_i is the output of the black box on the input $\beta_i=(\alpha_1^i,\ldots,\alpha_n^i)$ hence we have

$$v_i = a_1 r_1^i + a_2 r_2^i + \cdots + a_t r_t^i$$

for $0 \le i \le 2t-1$. Note that the system of equations obtained from the above set of equations is a Vandermonde system which can be solved in $O(t^2)$ time and $O(t)$ space (See [16]). After solving this system of equations we get

$$a_1=1, a_2=91, a_3=42, a_4=61 \quad \text{and} \quad a_5=94$$

and hence $f = 1 + 91yz^2 + 42z^5 + 61x^2y^2z + 94x^2yz$ is interpolated and we are done.

3. PROBLEMS

The evaluation points $\alpha_1,\ldots,\alpha_n,\alpha_{n+1}$ must satisfy certain conditions for our new algorithm to work properly. Here we identify all problems.

Distinct Monomials

The first condition is that for $i \neq j$

$$M_i(\alpha_1,\ldots,\alpha_n) \neq M_j(\alpha_1,\ldots,\alpha_n) \ \text{ in } \mathbb{Z}_p.$$

Also at the k'th step of the algorithm, when computing the degrees of the monomials in x_k, we must have

$$\forall\ 1 \le i \neq j \le t \Rightarrow m_i^k \neq m_j^k, \ \text{ in } \mathbb{Z}_p$$

where $m_i^k = M_i(\alpha_1,\ldots,\alpha_{k-1},\alpha_{n+1},\alpha_{k+1},\ldots,\alpha_n)$. If $m_i^k = m_j^k$ in $\mathbb{Z}_p$ then $\deg_z(\Lambda_{k+1}) < t$ and hence Λ_{k+1} will not have t distinct linear factors. To reduce the probability of monomial evaluations colliding, we should pick α_i to be distinct and to have order $> d$. The easiest way to do this is to use generators. There are $\phi(p-1)$ generators in $\mathbb{Z}_p$ where ϕ is Euler's function.

We now compute an upper bound on the probability that $M_i(\alpha_1,\ldots,\alpha_n) \neq M_j(\alpha_1,\ldots,\alpha_n)$ for all $1 \le i \neq j \le t$, for a set of random evaluation points $\{\alpha_1,\ldots,\alpha_n\}$, where α_i's are distinct generators of $\mathbb{Z}_p^*$.

Lemma 1 *Let M_i and M_j be two distinct monomials in $x_1,\ldots,x_n$. If $\alpha_1,\alpha_2,\ldots,\alpha_n$ are distinct generators of $\mathbb{Z}_p^*$, chosen at random, then in $\mathbb{Z}_p$*

$$\text{Prob}(M_i(\alpha_1,...,\alpha_n) = M_j(\alpha_1,...,\alpha_n)) \le \frac{d}{\phi(p-1)-n+1}.$$

PROOF. We will give a proof for $n=3$. Suppose $M_i = x^a y^b z^c$ and $M_j = x^k y^l z^m$. Since $M_i \neq M_j$, their degree must differ in at least one variable, say y so that $b \neq l$. Since α_1 is a generator, distinct from α_2 and α_3, we have $\alpha_2 = \alpha_1^s$ and $\alpha_3 = \alpha_1^t$ for some $1 < s,t < p-1$ and $s \neq t$ thus if

$$\alpha_1^a \alpha_2^b \alpha_3^c \equiv \alpha_1^k \alpha_2^l \alpha_3^m \pmod p$$

then

$$\alpha_1^{a-k} \alpha_1^{s(b-l)} \alpha_1^{t(c-m)} \equiv 1 \pmod p.$$

Since α_1 has order $p-1$ this implies

$$(a-k) + s(b-l) + t(c-m) \equiv 0 \pmod{p-1}.$$

For any t, the number of solutions to this equation for s is at most $g = \gcd(b-l,(a-k)+t(c-m),p-1)$. Since $|a-k|, |b-l|$ and $|c-m|$ are all bounded by d, and $b-l \neq 0$, we have $g \le d$. The maximum number of solutions occurs, for example, when $M_i = x^d y^0 z^c$ and $M_j = x^0 y^d z^c$ and $d|(p-1)$. Since α_2 is a generator distinct from α_1 and α_3, we have $\gcd(s,p-1)=1$ and $1 \neq s \neq t$. Thus there are $\phi(p-1)-2$ choices for s and we obtain

$$\text{Prob}(M_i(\alpha_1,\ldots,\alpha_n) = M_j(\alpha_1,\ldots,\alpha_n)) \ \le \ \frac{d}{\phi(p-1)-2}.$$

□

Remark 1 Note that this upper bound is pessimistic. In applications where the prime p can be chosen, if one chooses p with $q=(p-1)/2$ also prime and $q > d$, which will often be easy to do, then $g \le 2$ and $\phi(p-1) = q-1$ and hence the bound $\frac{d}{\phi(p-1)-2}$ reduces to $\frac{2}{q-1-n+1} = \frac{4}{p-1-2n} \in O(1/p)$.

Theorem 1 *Since there are $\binom{t}{2}$ pairs of monomials M_i and M_j, the probability that no two evaluate to the same value is at most*

$$\frac{t(t-1)}{2} \cdot \frac{d}{\phi(p-1)-n+1} < \frac{dt^2}{\phi(p-1)}.$$

Root Clashing

Let $r_1,\ldots,r_t$ be the roots of $\Lambda_1(z)$ which is the output of the Berlekamp/Massey algorithm on the sequence of the outputs from the black box on the first set of evaluation points $\alpha_1,\ldots,\alpha_n$. Suppose at the k'th step, we want to compute the degrees of all the monomials in the variable x_k. As mentioned in the Example 1, the first step is to compute Λ_{k+1}. Then if $\deg_{x_k}(M_i) = e_{ik}$ we have $\bar{r}_i = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$ is a root of Λ_{k+1}. If $r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e'}$, $0 \le e' \neq e_{ik} \le d$ is also a root of Λ_{k+1} then we may not be able to uniquely identify the correct degree of the i'th monomial in the k'th variable x_k. We will illustrate this with an example.

Example 2 *Consider the polynomial given in Example 1. Suppose instead of choosing $\alpha_4 = 34$, we choose $\alpha_4 = 72$ which is another generator of $\mathbb{Z}_p^*$. Since α_1,α_2 and α_3 are the same as before, Λ_1 does not change and hence the roots of Λ_1 are $r_1=1, r_2=7, r_3=41, r_4=61$ and $r_5=64$. In the next step we substitute $\alpha_4=72$ for α_1 and compute $\Lambda_2 = z^5 + 61z^4 + 39z^3 + 67z^2 + 37z + 98$. We proceed to compute the degrees of the monomials in x but we find that both*

$$r_4 \times (\frac{\alpha_4}{\alpha_1})^2 = 15 \quad \text{and} \quad r_4 \times (\frac{\alpha_4}{\alpha_1})^4 = 7$$

are roots of Λ_2 and hence we can not decide the correct degree of the last monomial in x.

Theorem 2 *The probability that we would not be able to uniquely compute all the degrees in one variable x_k ($1 \le k \le n$) is approximately $\frac{3dt^2}{2p}$.*

PROOF. Let $S_i = \{r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^i \mid 1 \le j \le t\}$ for $0 \le i \le d$. We assume that $r_i \neq r_j$ for all $1 \le i \neq j \le t$. We will not be able to uniquely identify the degree of the j'th monomial in x_k if there exists $\bar{d}$ such that $r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} = \bar{r}_i$ is a root of Λ_{k+1} ($\bar{r}_i$) and $0 \le d \neq e_{jk} \le d$ where e_{jk} is $\deg_{x_k}(M_j)$. But we have $\bar{r}_i = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$ thus $r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} = r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{e_{ik}}$. With out loss of generality, assume $\bar{d} = \bar{d} - e_{ik} >$

0. We have $r_i = r_j \times (\frac{\alpha_{n+1}}{\alpha_k})^{\tilde{d}}$ and hence $r_i \in S_{\tilde{d}} \Rightarrow S_0 \cap S_{\tilde{d}} \neq \emptyset$. Hence we will not be able to compute the degrees in x_k if $S_0 \cap S_i \neq \emptyset$ for some $1 \leq i \leq d$. Assuming that $\frac{\alpha_{n+1}}{\alpha_k}$ has a sufficiently high order in $\mathbb{Z}_p$ (at least $d+1$), we can assume that the elements in S_i are random elements in $\mathbb{Z}_p$. Hence the probability that $S_0 \cap S_i \neq \emptyset$ is

$$1 - \prod_{i=1}^{t} (1 - \frac{t+i}{p}) \approx 1 - \mathrm{e}^{\frac{-t(3t+1)}{2p}}.$$

If we sum this quantity for all $1 \leq i \leq d$ we obtain that the overall probability is approximately $d \times (1 - \mathrm{e}^{\frac{-t(3t+1)}{2p}})$. For $t^2 \ll p$ we have $1 - \mathrm{e}^{\frac{-t(3t+1)}{2p}} \approx \frac{3t^2}{2p}$. This completes the proof. □

Using Theorem 2, the probability that we will not be able to uniquely identify the degrees of the monomials in *all* the variables is approximately $\frac{3ndt^2}{2p}$, i.e. for $p \approx 3ndt^2$ with probability about half, the algorithm succeeds without dealing with any problem. We will now discuss our solution to this problem. Note that we assume the images of the monomials are distinct, i.e. $\forall\ 1 \leq i \neq j \leq t \Rightarrow m_i^k \neq m_j^k$. Suppose we have computed Λ_{k+1} and we want to compute the degrees of the monomials in x_k and let $R_1 = \{r_1, \ldots, r_t\}$ be the set of all the roots of Λ_1 and $R_k = \{\bar{r}_1, \ldots, \bar{r}_t\}$ be the set of all the distinct roots of Λ_{k+1}. Let

$$D_j = \{(i, r) \mid 0 \leq i \leq d,\ r = r_j \times (\frac{\alpha_{n+1}}{\alpha_1})^i \in R_k\}.$$

D_j contains the set of all possible degrees of the j'th monomial M_j in the k'th variable x_k. We know that $(e_{jk}, \bar{r}_j) \in D_j$ and hence $|D_j| \geq 1$. If $|D_j| = 1$ for all $1 \leq j \leq t$, then the degrees are unique and this step of the algorithm is complete. Let G_k be a balanced bipartite graph defined as follows. G_k has two independent sets of nodes U and V each of size t. Nodes in U and V represent elements in R_1 and R_k respectively, i.e. $u_i \in U$ and $v_j \in V$ are labelled with r_i and $\bar{r}_j$. We connect $u_i \in U$ to $v_j \in V$ with an edge of weight (degree) d_{ij} if and only if $(d_{ij}, \bar{r}_j) \in D_i$.

Lemma 2 *We can uniquely identify the degrees of all the monomials in x_k, if and only if the bipartite graph G_k has a unique* perfect matching.

The proof of this lemma is immediate by looking at the structure of the graph G_k. We illustrate this with an example.

Example 3 *Let f be the polynomial given in Example 1 and suppose for some evaluation points $\alpha_1, \ldots, \alpha_4$ we obtain the graph G_1 as shown in Figure 1. This graph has a perfect matching, i.e. the set of edges $\{(r_i, \bar{r}_i) \mid 1 \leq i \leq 5\}$. If there was an edge connecting r_1 to $\bar{r}_2$ then the new graph would no longer have a unique perfect matching and we would fail to uniquely compute the degrees of monomials in x.*

We now give a solution for the case where G_k does not have a unique perfect matching for some $1 \leq k \leq n$. The solution involves $2t$ more probes to the black box. Suppose we choose a random element $\alpha_{n+2} \in \mathbb{Z}_p$ such that $\gamma = \frac{\alpha_{n+2}}{\alpha_{n+1}}$ is a generator of $\mathbb{Z}_p^*$ (or is of order greater than d). Let $\beta_i = (\alpha_1^i, \ldots, \alpha_{k-1}^i, \alpha_{n+2}^i, \alpha_{k+1}^i, \ldots, \alpha_n^i)$ and let v_i be the output of the black box on input β_i $(0 \leq i \leq 2t-1)$. On input $V = (\beta_0, \ldots, \beta_{2t-1})$, the Berlekamp/Massey algorithm

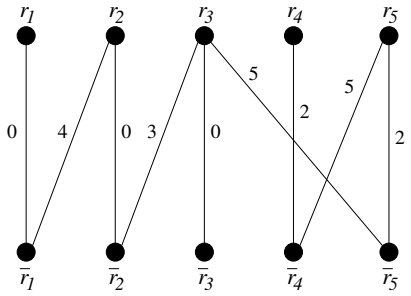

Figure 1: The bipartite graph G_1

computes a linear generator $\Lambda'_{k+1}(z)$ for V. Let $\{\tilde{r}_1, \ldots, \tilde{r}_t\}$ be the set of distinct roots of Λ'_{k+1}. Let G'_k be the balanced bipartite graph, obtained from Λ_1 and Λ'_{k+1}.

Definition 1. We define $\bar{G}_k$, the intersection of G'_k and G_k, as follows. $\bar{G}_k$ has the same nodes as G'_k and there is an edge between r_i and $\tilde{r}_j$ with weight (degree) d_{ij} if and only if r_i is connected to $\bar{r}_j$ in G_k and to $\tilde{r}_j$ in G'_k, both with the same degree d_{ij}.

Lemma 3 *Let $e_{ij} = \deg_{x_j}(M_i)$. The two nodes r_i and $\tilde{r}_i$ are connected in $\bar{G}_k$ with degree e_{ij}.*

We take advantage of the following theorem which implies we need at most one extra set of probes.

Theorem 3 *Let $\bar{G}_k = G_k \cap G'_k$. $\bar{G}_k$ has a* unique *perfect matching.*

Proof. Let U and V be the set of independent nodes in $\bar{G}_k$ such that $u_i \in U$ and $v_j \in V$ are labelled with r_i and $\tilde{r}_j$ respectively where $\tilde{r}_j$ is a root of Λ'_{k+1}. We will prove that each node in V has degree exactly 1 and hence there is a unique perfect matching. The proof is by contradiction. Suppose the degree of $v_j \in V$ is at least 2. With out loss of generality assume that r_1 and r_2 are both connected to $\tilde{r}_j$ with degrees d_{1j} and d_{2j} respectively (See Figure 2).

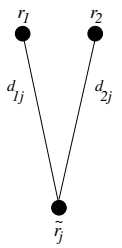

Figure 2: Node $\tilde{r}_j$ of graph $\bar{G}_k$

Using Definition 1 we have

$$\bar{r}_j = r_1 \times (\frac{\alpha_{n+1}}{\alpha_k})^{d_{1j}} = r_2 \times (\frac{\alpha_{n+1}}{\alpha_k})^{d_{2j}} \quad \text{and}$$
$$\tilde{r}_j = r_1 \times (\frac{\alpha_{n+2}}{\alpha_k})^{d_{1j}} = r_2 \times (\frac{\alpha_{n+2}}{\alpha_k})^{d_{2j}}.$$

Dividing the two sides of these equations results in

$$(\frac{\alpha_{n+2}}{\alpha_{n+1}})^{d_{1j}} = (\frac{\alpha_{n+2}}{\alpha_{n+1}})^{d_{2j}}.$$

Since we chose α_{n+2} such that $\frac{\alpha_{n+2}}{\alpha_{n+1}}$ has a sufficiently large order (greater than the degree bound d) we have $d_{1j} = d_{2j} \Rightarrow r_1 = r_2$. But this is a contradiction because both r_1 and r_2 are roots of Λ_1 which we assumed are distinct. □

Lemma 3 and Theorem 3 prove that the intersection of G_k and G'_k will give us the correct degrees of all the monomials in the k'th variable x_k. We will illustrate with an example.

Example 4 *Let* $f = -10\,y^3 - 7\,x^2yz - 40\,yz^5 + 42\,y^3z^5 - 50\,x^7z^2 + 23\,x^5z^4 + 75\,x^7yz^2 - 92\,x^6y^3z + 6\,x^3y^5z^2 + 74\,xyz^8 + 4$ *and* $p = 101$. *We choose the first set of evaluation points to be* $\alpha_1 = 66, \alpha_2 = 11, \alpha_3 = 48$ *and* $\alpha_4 = 50$. *For the first variable* x *we will obtain the bipartite graph* G_1 *shown in Figure 3.*

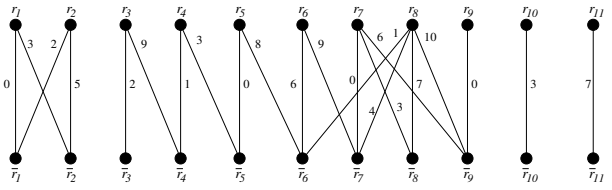

Figure 3: The bipartite graph G_1

This graph does not have a unique perfect matching, so we proceed to choose a new evaluation point $\alpha_5 = 89$. *This time we will get the bipartite graph* G'_1 *shown in Figure 4.*

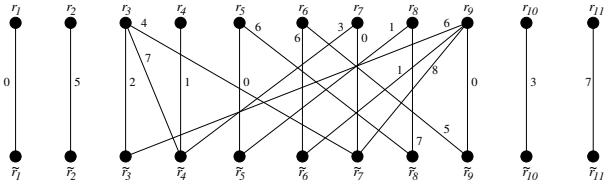

Figure 4: The bipartite graph G'_1

Again G'_1 *does not have a unique perfect matching. We compute the intersection of* G_1 *and* G'_1*:* $\bar{G}_1 = G_1 \cap G'_1$. $\bar{G}_1$ *is shown in Figure 5.*

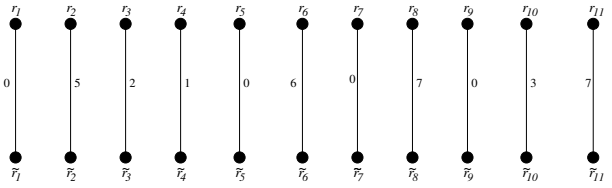

Figure 5: The bipartite graph $\bar{G}_1$

As stated by Theorem 3, $\bar{G}_1$ *has a unique perfect matching and the degrees of every monomials in* x *is correctly computed.*

In this section we proved that if the prime p is sufficiently large ($\phi(p-1)$ must be approximately dt^2 for us to be able to get distinct images of monomials with reasonable probability), we will be able to compute the degrees of all the t monomials in each variable x_k using up to $4t$ evaluation points. If the graph G_k has a unique perfect matching, we will be able to compute the degrees in x_k with only $2t$ probes to the black box.

We conclude this section with the following lemma which we will later use in Section 4.

Lemma 4 *Let* G_k *be the bipartite graph for the* k*'th variable. Let* $u_{i_1} \to v_{j_1} \to u_{i_2} \to v_{j_2} \to \cdots \to v_{j_s} \to u_{i_1}$ *be a cycle in* G_k *where* $u_l \in U$ *is labelled with* r_l *(a root of* Λ_1*) and* $v_m \in V$ *is labelled with* $\bar{r}_m$ *(a root of* Λ_{k+1}*). Let* d_{lm} *be the weight (degree) of the edge between* u_l *and* v_m. *We have* $\sum_{m=1}^{s} d_{i_m j_m} - \sum_{m=1}^{s} d_{i_{m+1} j_m} = 0$.

Proof. It is easy to show that $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}} r_{i_s}$ where $\bar{d} = d_{i_1 j_1} - d_{i_2 j_1} + d_{i_2 j_2} - d_{i_3 j_2} + \cdots + d_{i_s j_{s-1}} - d_{i_s j_{s-1}}$. Also both u_{i_1} and u_{i_s} are connected to v_{j_s} in G_k hence we have $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{d_{i_1 j_s}} \bar{r}_{i_s}$ and $r_{i_s} = (\frac{\alpha_{n+1}}{\alpha_k})^{d_{i_s j_s}} \bar{r}_{i_s}$. These three equations yield to $r_{i_1} = (\frac{\alpha_{n+1}}{\alpha_k})^{\tilde{d}} r_{i_1}$ where $\tilde{d} = d_{i_1 j_1} - d_{i_2 j_1} + d_{i_2 j_2} - d_{i_3 j_2} + \cdots + d_{i_{s-1} j_{s-1}} - d_{i_s j_{s-1}} + d_{i_s j_s} - d_{i_1 j_s}$. But if $\frac{\alpha_{n+1}}{\alpha_k}$ is of sufficiently high order, $\tilde{d}$ must be zero thus $\sum_{m=1}^{s} d_{i_m j_m} - \sum_{m=1}^{s} d_{i_{m+1} j_m} = 0$. □

Example 5 *In the graph* G'_1 *shown in Figure 4, there is a cycle* $r_3 \to \tilde{r}_4 \to r_7 \to \tilde{r}_7 \to r_3$. *The weights (degrees) of the edges in this cycle are as* $7, 3, 0$ *and* 4 *respectively. We have* $7 - 3 + 0 - 4 = 0$.

4. THE ALGORITHM

Algorithm: Interpolation

Input:

- A *black box* $\mathbb{Z}_p^n \to \mathbb{Z}_p$ that on input $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p^n$ outputs $f(\alpha_1, \ldots, \alpha_n)$ where $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ is the target polynomial.
- A degree bound $d \in \mathbb{Z}$ such that $\deg_{x_i}(f) \leq d$ for any $1 \leq i \leq n$. Also $\phi(p-1) > dt^2 + n$ (*See Theorem 1*).
- A bound $T \geq t$ on the number of terms in f.

Output: The polynomial f.

1: Factor $p-1$ and let Q be the set of all the prime divisors.
2: For $1 \leq i \leq n+1$, choose distinct $\alpha_i \in \mathbb{Z}_p$ at random s.t. $\alpha_i^{\frac{p-1}{q_j}} \neq 1 \bmod p$, for all $q_j \in Q$. α_i is a random primitive $(p-1)$'th root of unity.
3: Choose γ to be a random generator of $\mathbb{Z}_p^*$ and let $\alpha_{n+2} = \alpha_{n+1} \times \gamma$. Repeat until $\alpha_{n+2} \notin \{\alpha_1, \ldots, \alpha_{n+1}\}$.
4: Let $\beta_i = (\alpha_1^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2T-1$ and let v_i be the output of the black box on input v_i.
5: Use the Berlekamp/Massey algorithm to compute a generator $\Lambda_1(z)$ for $v_0, \ldots, v_{2T-1}$ and set $t = \deg_z(\Lambda_1)$.
6: Let $\{r_1, \ldots, r_t\}$ be the set of distinct roots of $\Lambda_1(z)$.
7: **for** k from 1 to n **do**
8: Determine $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$:
9: Let $\beta_i = (\alpha_1^i, \ldots, \alpha_{k-1}^i, \alpha_{n+1}^i, \alpha_{k+1}^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2t+1$ and let v_i be the output of the black box on input β_i. (*Substitute* α_k *by* α_{n+1}).
10: Use the Berlekamp/Massey algorithm to compute a linear generator Λ_{k+1} for the sequence $v_0, \ldots, v_{2t+1}$.
11: If $\deg_z(\Lambda_{k+1}) < t$ then the monomial evaluations are not all distinct so choose a new generator α_{n+1} for $\mathbb{Z}_p^*$ and go back to Step 7.

12: If $\deg_z(\Lambda_{k+1}) > t$ then the monomial evaluations used to compute $\Lambda_1(z)$ were not all distinct so t is wrong so go back to Step 2 and restart.
13: Construct the bipartite graph G_k as described in Section 3.
14: **if** G_k has a unique perfect matching **then**
15: Set $e_{ik} = d_{il}$ where d_{il} is the weight (degree) of the edge that matches the node r_i to $\bar{r}_l$ in the perfect matching. For $1 \leq i \leq t$, $e_{ik} = \deg_{x_k}(M_i)$.
16: **else**
17: Repeat Steps 7 and 12 but use α_{n+2} instead of α_{n+1} to obtain Λ'_{k+1}.
18: Construct the bipartite graph G'_k as described in Section 3.
19: Find the intersection of G_k and G'_k: $\bar{G}_k = G_k \cap G'_k$.
20: Set $e_{ik} = d_{il}$ where d_{il} is the weight (degree) of the edge that matches the node r_i to $\tilde{r}_l$ in the perfect matching of graph $\bar{G}_k$. For $1 \leq i \leq t$, $e_{ik} = \deg_{x_k}(M_i)$.
21: **end if**
22: **end for**
23: Let $S = \{a_1 r_1^i + a_2 r_2^i + \cdots + a_t r_t^i = v_i \mid 0 \leq i \leq 2t-1\}$ be a *Vandermonde* system of linear equations. Solve this to obtain $a_1, \ldots, a_t$. ($a_i \in \mathbb{Z}_p$ *is the coefficient of* M_i)
24: Output $f = \sum_{i=1}^{t} a_i M_i$ where $M_i = \prod_{j=1}^{n} x_j^{e_{ij}}$.

Remark 2 In Step 4, if monomial evaluations collide, the degree of $\Lambda_1(z)$ will be less than the number of terms of f. To detect this, each time we apply the Berlekamp/Massey algorithm in Step 10, instead of using $2t$ evaluation points, we use $2t+2$ probes to detect if $\Lambda_1(z)$ has the right degree with high probability.

Remark 3 We use the version of the Berlekamp/Massey algorithm given by Kaltofen *et al.* in [8]. We believe there is a minor error in the algorithm presented in [8]. The variable B_0 must be initialized to one rather than zero.

4.1 Complexity Analysis

We now discuss the complexity of the algorithm presented in Section 4. To compute the roots of $\Lambda_1(z)$ at Step 6 of the algorithm, we use Rabin's probabilistic algorithm [13]. Rabin's algorithm tries to split the polynomial of degree t into two polynomials with smaller degrees at each step by computing the $\gcd((z-\beta)^{(p-1)/2} - 1, \Lambda_1(z))$ for randomly chosen $\beta \in \mathbb{Z}_p$. Since $\deg_z(\Lambda_1) = t$, the cost of finding the t roots of $\Lambda_1(z)$, assuming classical algorithms for polynomial arithmetic in $\mathbb{Z}_p[z]$ are used, is $O(t^2 \log p)$ (See Algorithm 14.15 of [14]).

The Vandermonde system of equations at Step 23 can be solved in $O(t^2)$ using the technique given in [16]. Note that as mentioned in [16], when inverting a $t \times t$ Vandemonde matrix defined by $k_1, \ldots, k_t$, one of the most expensive parts of this technique is to compute the master polynomial $M(z) = \prod_{i=1}^{t}(z - k_i)$. However, in our algorithm we can use the fact that $M(z) = \prod_{i=1}^{t}(z - r_i) = \Lambda_1(z)$. Also the Berlekamp/Massey algorithm (as presented in [8]) runs in $O(t^2)$ time.

We can compute the information needed to construct the bipartite graph G_k in $O(dt^2)$ time. This involves evaluating $\Lambda_{k+1}(z)$ at d points for each monomial and testing if it is zero or not. Also computing the intersection of G_k and G'_k can be done in $O(td \log d)$ time. This is because we know that each node in the intersection is of degree one (See proof of Theorem 3). Hence, the total time for running the loop in Step 7 of the algorithm is $O(ndt^2 + nt^2 + td \log d) = O(ndt^2 + td \log d)$.

We also need to consider the cost of probing the black box. Let $E(n,t,d)$ be the cost of one probe to the black box. If G_k has a unique perfect matching for $1 \leq k \leq n$ then we can correctly compute the degrees using only G_k. In this case the total number of evaluation points used is exactly $2(n+1)t$. In the worst case where G_k does not have a unique perfect matching for all $1 \leq k \leq n$, we need to do additional $2nt$ probes to the black box to construct all $G'k$ graphs. In this case the total number of probes to the black box is $2(n+1)t + 2nt = 2(2n+1)t$. In both cases the number of probes is $N_s = O(nt)$ and hence the total cost of probes to the black box is $O(ntE(n,t,d))$. Thus the total cost of running the algorithm is in $O(t^2(\log(p) + nd) + ntE(n,t,d))$. Assuming $\log(p) \in O(nd)$ (this is true if $p \in O(ndt^2)$ because $t \leq (d+1)^n$), the total cost is in $O(ndt^2 + ntE(n,t,d))$.

Zippel's Algorithm

For comparison, we will briefly discuss the complexity of Zippel's 1990 interpolation algorithm. Let t_i be the number of terms in the target polynomial f after evaluating variables $x_{i+1}, \ldots, x_n$. We have $t_0 = 1$ and $t_n = t$. The number of probes to the black box using Zippel's algorithm is

$$N_z = d + 1 + (t_1 d + t_2 d + \ldots t_{n-1} d) = 1 + d \sum_{i=0}^{n-1} t_i.$$

Since $t_i \leq t_n$ for all $1 \leq i \leq n-1$, the number of probes is in $O(ndt)$. Also Zippel's interpolation algorithm does $O(ndt^2)$ operations so the total cost is in $O(ndt^2 + ndtE(n,t,d))$.

Example 6 *Let* $f = x^{20} + y^{20} + z^{20} + 1$ *with* $p = 1009, d = 20$ *and* $t = 4$. *Our new algorithm will do* $N_s = 32$ *probes to the black box while Zippel's does about* $N_z = 121$ *probes. Also note that a bad degree bound d will not affect the number of probes to the black box in our new algorithm but this number will linearly increase in Zippel's algorithm. As an example if choose the bound* $d = 40$ *instead of* $d = 20$, *the number of probes in Zippel's algorithm will almost double:* $N_z = 241$.

We expect Zippel's algorithm to perform better than our algorithm for *dense* target polynomials.

Lemma 5 *Let* f *be a completely dense polynomial such that* $\deg_{x_i}(f) = d$ *(total degree of* f *is* nd*). The number of terms in* f *is* $t = (d+1)^n$. *The number of probes to the black box in Zippel's algorithm is exactly* $N_z = t$.

PROOF. Here we have $t_i = (d+1)^i$ thus $N_z = 1 + d \times \prod_{i=0}^{n-1}(d+1)^i = 1 + d \times \frac{(d+1)^n - 1}{d+1-1} = (d+1)^n = t$. □

In this case, our algorithm does at most $4(n+1)t$ probes.

4.2 Optimizations

Let $d_{max} = \max\{\deg_{x_i}(f) \mid 1 \leq i \leq n\}$. If the prime p is large enough, i.e. $p > \frac{3nd_{max}t^2}{2\epsilon}$ then with probability $1 - \epsilon$ the degree of every monomial in x_k can correctly be computed using only G_k and without needing any extra probes to the black box. In fact in this case, with high probability,

every r_i will be matched with exactly only one $\bar{r}_j$ and hence every node in G_k would have degree one (e.g. see Figure 5). But if $d \gg d_{max}$, i.e. the degree bound d is not tight, the probability that we could identify the degrees uniquely drops significantly even though p is large enough. This is because the probability that *root clashing* (see Section 3) happens, linearly depends on d. In this case, with probability $1-\epsilon$, the degree of M_i in x_k would be $\min\{d_{ij} \mid (d_{ij}, r_i) \in G_k\}$, i.e. the edge connected to r_i in G_k with minimum weight (degree) is our desired edge in the graph which will show up in the prefect matching.

We can also use the following theorem.

Theorem 4 *Let H_k be a graph obtained by eliminating all edges connected to r_i in G_k except the one with minimum weight (degree) for all $1 \le i \le t$. If the degree of every node in H_k is exactly one, then e_{ik} is equal to the weight of the edge connected to r_i in H_k.*

This theorem can be proved using Lemma 4 and the fact that there can not be any cycle in the graph H_k. We will give an example.

Example 7 *Let $f = 25y^2z+90yz^2+93x^2y^2z+60y^4z+42z^5$. Here $t = 5, n = 3, d_{max} = 5$ and $p = 101$. We choose the following evaluation points $\alpha_1 = 85, \alpha_2 = 96, \alpha_3 = 58$ and $\alpha_4 = 99$. Suppose we want to construct G_2 in order to compute the degrees of the monomials in y. Suppose our degree bound is $d = 40$ which is not tight. The graph G_2 and H_2 are shown in Figures 6 and 7 respectively.*

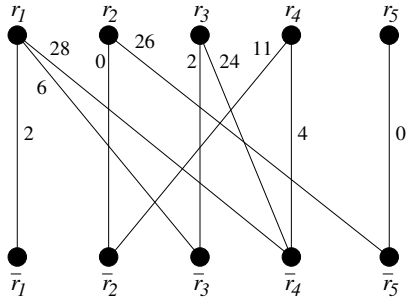

Figure 6: The bipartite graph G_2

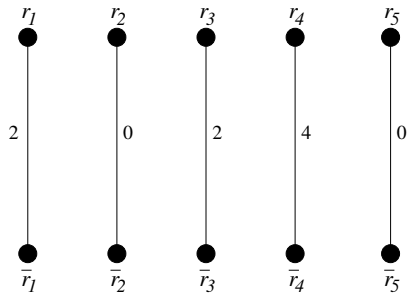

Figure 7: The bipartite graph H_2

The graph H_2 has the correct degrees of the monomials in variable y.

Theorem 4 suggests the following optimization. In the construction of the bipartite graph G_k, connect r_i to $\bar{r}_j$ with degree d_{ij} *only if* there is no $\bar{d} < d_{ij}$ such that $r_i \times (\frac{\alpha_{n+1}}{\alpha_k})^{\bar{d}}$ is a root of Λ_{k+1}, i.e. the degree of the node r_i in U is always one for all $1 \le i \le n$. If there is a perfect matching in this graph, this perfect matching is unique because this implies that the degree of each node $\bar{r}_j$ in V is also one (e.g. see Figure 7). If not, go back to and complete the graph G_k.

The second optimization is to compute the degree of each monomial $M_i = x_1^{e_{i1}} x_2^{e_{i2}} ... x_n^{e_{in}}$ in the last variable x_n without doing any more probes to the black box. Suppose we have computed the degree of M_i in x_k for $1 \le k < n$. We know that $M_i(\alpha_1, \ldots, \alpha_n)$ is equal to r_i, a root of Λ_1. Hence $r_i = \alpha_1^{e_{i1}} \cdot \alpha_2^{e_{i2}} \cdot \cdots \cdot \alpha_n^{e_{in}}$. Since we know the degrees e_{ij} for $1 \le j < n$ we can determine e_{in} by division by α_n. This reduces the total number of probes from $4(n+1)t$ to $4nt$.

5. BENCHMARKS

We have implemented both Zippel's sparse interpolation algorithm and our new algorithm in C programming language. We have also implemented an interface to call the interpolation routines from Maple. In this section we will give benchmarks comparing the performances on five problem sets. The polynomials in the first three sets have $n = 3$ variables while those in the last two sets have $n = 6$ and $n = 12$ variables respectively.

We count the number of probes to the black box and measure the total CPU time. All the timings given in this section are in CPU seconds and are obtained using Maple 13 on a 64 bit Intel Core i7 920 @ 2.66GHz, running Linux. The black box in our benchmarks computes a multivariate polynomial with coefficients in $\mathbb{Z}_p$ where $p = 3037000453$ is a 31.5 bit prime. In all these benchmarks, the black box simply evaluates the polynomial at the given evaluation point. To evaluate efficiently we pre-compute and cache the values of x_i^j mod p in a loop for $0 \le j \le d$ for each variable in $O(nd)$. Then we evaluate the t terms in $O(nt)$. Hence the cost of one black box probe is $O(nd + nt)$ arithmetic operations in $\mathbb{Z}_p$.

Benchmark #1

This set of problems consists of 13 multivariate polynomials in $n = 3$ variables. The i'th polynomial ($1 \le i \le 13$) is generated at random using the following command in Maple:

```
> randpoly([x1,x2,x3], terms = 2^i, degree = 30) mod p;
```

The i'th polynomial will have about 2^i non-zero terms. Here $D = 30$ is the total degree hence the maximum number of terms in each polynomial is $t_{max} = \binom{n+D}{D} = 5456$. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 30$. The timings and the number of probes are given in Table 1.

Note that the last polynomial $i = 13$, is almost completely dense. The data in Table 1 shows that for sparse polynomials $1 \le i \le 6$, our new algorithm does a lot less probes to the black box compared to Zippel's algorithm. It is also faster than Zippel's algorithm for $1 \le i \le 9$. However, as the polynomials get denser, Zippel's algorithm has a better performance. For a completely dense polynomial with t nonzero terms, Zippel's algorithm only does $O(t)$ probes to the black box while the new algorithm does $O(nt)$ probes, a factor of n more. This is clearly shown in the last row of Table 1.

As i increases, the polynomial f becomes denser. For $i > 6$, f has more than $\sqrt{t_{max}}$ non-zero terms which means

i	$\#f$	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00	15	0.01	961
2	4	0.00	27	0.01	558
3	8	0.00	51	0.01	1209
4	16	0.00	99	0.01	1054
5	32	0.00	195	0.01	1612
6	64	0.01	387	0.03	2728
7	128	0.03	771	0.09	4433
8	253	0.11	1521	0.21	6603
9	512	0.43	3075	0.55	9765
10	1015	1.64	6093	1.15	12431
11	2041	6.42	12249	2.42	15128
12	4081	24.96	24489	4.56	16182
13	5430	43.67	32583	5.92	16430

Table 1: Timings and number of probes for the first set of problems.

it is becoming dense. This is indicated by a horizontal line in Table 1 and also in the subsequent benchmarks.

To show how effective the first optimization described in Section 4.2 is, we run both our algorithm and Zippel's algorithm on the same set of polynomials but with a bad degree bound $d = 100$. The timings and the number of probes are given in Table 2.

i	$\#f$	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00	15	0.04	3131
2	4	0.00	27	0.02	1818
3	8	0.00	51	0.04	3939
4	16	0.00	99	0.04	3434
5	32	0.00	195	0.07	5252
6	64	0.01	387	0.15	8888
7	128	0.04	771	0.36	14433
8	253	0.12	1521	0.80	21513
9	512	0.45	3075	1.96	31815
10	1015	1.66	6093	3.98	40501
11	2041	6.48	12249	8.16	49288
12	4081	25.05	24489	15.14	52722
13	5430	43.77	32583	19.61	53530

Table 2: Timings and number of probes for the first set of problems with bad degree bound $d = 100$.

Benchmark #2

In this set of benchmarks the i'th polynomial is in three variables and generated at random using the following command in Maple:

```
> randpoly([x1,x2,x3], terms = 2^i, degree = 100) mod p;
```

This set of polynomials is similar to polynomials in the first benchmark except that the total degree of each polynomial is set to be 100 in the second set. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 100$. The timings and the number of probes are given in Table 3.

i	$\#f$	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00	15	0.03	2929
2	4	0.00	27	0.04	3131
3	8	0.00	51	0.09	6363
4	16	0.00	99	0.09	6767
5	31	0.01	189	0.19	11514
6	64	0.01	387	0.29	14948
7	127	0.05	765	0.55	21008
8	253	0.17	1521	1.47	37744
9	511	0.64	3069	4.42	58479
10	1017	2.50	6105	14.00	99384
11	2037	9.69	12225	43.28	167155
12	4076	38.17	24489	121.54	263105
13	8147	150.00	48885	282.54	359863
14	16282	590.81	97695	589.82	442178

Table 3: Timings and number of probes for the second set of problems.

Comparing this table to the data in Table 1 shows that the number of probes to the black box in our new algorithm does not depend on the degree of the target polynomial whereas it does for Zippel's algorithm.

Benchmark #3

This set of problems consists of 14 multivariate polynomials in $n = 6$ variables. The i'th polynomial ($1 \leq i \leq 13$) is generated at random using the following command in Maple:

```
>randpoly([seq(x||j,j=1..6)],terms=2^i,degree=30) mod p;
```

The i'th polynomial will have about 2^i non-zero terms. Here $D = 30$ is the total degree. We run both the Zippel's algorithm and our new algorithm with degree bound $d = 30$. The timings and the number of probes are given in Table 4.

i	$\#f$	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00	30	0.01	744
2	3	0.00	42	0.01	961
3	8	0.00	102	0.01	1550
4	16	0.00	198	0.02	2697
5	31	0.00	378	0.05	4557
6	64	0.02	774	0.15	8246
7	127	0.06	1530	0.44	14632
8	255	0.21	3066	1.51	27714
9	511	0.81	6138	5.19	50685
10	1016	3.08	12198	17.92	91047
11	2037	12.10	24450	65.31	168330
12	4083	47.82	49002	230.44	301382
13	8151	187.94	97818	803.60	532580
14	16287	741.21	195450	> 1000	–

Table 4: Timings and number of probes for the third set of problems.

Benchmark #4

In this set of problems, the 14 polynomials are chosen at random in 12 variables with total degree $D = 10$.

```
>n := 12;
>randpoly([seq(x||j,j=1..n)],terms=2^i,degree=10) mod p;
```

We run both the Zippel's algorithm and our new algorithm with degree bound $d = 10$. The timings and the number of probes are given in Table 5.

i	$\#f$	New Algorithm		Zippel's Algorithm	
		Time	Probes	Time	Probes
1	2	0.00	60	0.00	396
2	3	0.00	108	0.01	605
3	8	0.00	204	0.01	1001
4	16	0.00	396	0.02	1760
5	32	0.01	780	0.04	3113
6	64	0.04	1548	0.14	6017
7	128	0.12	3036	0.47	11154
8	253	0.48	6132	1.57	19976
9	512	1.88	12252	5.81	37609
10	1016	7.28	24396	20.30	67364
11	2039	28.70	48804	73.61	123343
12	4075	114.39	97764	257.39	217987
13	8146	454.91	195516	916.81	391919
14	16284	1809.32	390828	3142.69	679294

Table 5: Timings and number of probes for the fourth set of problems.

Tables 1,3,4 and 5 show that our algorithm is more efficient compared to Zippel's sparse interpolation algorithm if the target polynomial is sparse. For denser polynomials Zippel's algorithm does less probes to the black box and hence is more efficient. Also the number of probes to the black box in our new algorithm is independent of the degree bound d. This is not true for Zippel's algorithm which depends linearly on d.

6. CONCLUSION

Our sparse interpolation algorithm is a modification of the Ben-Or/Tiwari algorithm [1] for polynomials over finite fields. It costs an extra factor of between n and $2n$ probes. Although we presented our algorithm for interpolating over $\mathbb{Z}_p$, it also works over an arbitrary finite field $\mathbb{F} = GF(q)$. Further, if p (or q) is too small, one can work inside a suitable extension field. Our benchmarks show that it in practice, it does fewer probes to the black box than Zippel's algorithm when the polynomial is sparse, e.g., when f has fewer than the square root of the maximum possible number of terms.

7. REFERENCES

[1] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. of the twentieth annual ACM symposium on Theory of computing*, pages 301–309. ACM, 1988.

[2] Jennifer de Kleine, Michael Monagan, and Allan Wittkopf. Algorithms for the non-monic case of the sparse modular gcd algorithm. In *Proceedings of ISSAC'05*, pages 124–131. ACM, 2005.

[3] Mark Giesbrecht, Erich Kaltofen, and Wen-shin Lee. Algorithms for computing the sparsest shifts of polynomials via the berlekamp/massey algorithm. In *Proceedings of ISSAC'02*, pages 101–108. ACM, 2002.

[4] Dima Yu. Grigoriev and Y. N. Lakshman. Algorithms for computing sparse shifts for multivariate polynomials. In *Proceedings of ISSAC'95*, pages 96–103. ACM, 1995.

[5] S. M. Mahdi Javadi and M. B. Monagan. A sparse modular gcd algorithm for polynomials over algebraic function fields. In *Proceedings of ISSAC '07*, pages 187–194. ACM, 2007.

[6] Erich Kaltofen and Yagati N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *Proceedings of ISSAC'88*, pages 467–474. Springer-Verlag, 1989.

[7] Erich Kaltofen and Wen-shin Lee. Early termination in sparse interpolation algorithms. *J. Symb. Comput.*, 36(3-4):365–400, 2003.

[8] Erich Kaltofen, Wen-shin Lee, and Austin A. Lobo. Early termination in ben-or/tiwari sparse interpolation and a hybrid of zippel's algorithm. In *Proceedings of ISSAC'00*, pages 192–201. ACM, 2000.

[9] Y. N. Lakshman and B. David Saunders. Sparse polynomial interpolation in nonstandard bases. *SIAM J. Comput.*, 24(2):387–397, 1995.

[10] Y. N. Lakshman and B. David Saunders. Sparse shifts for univariate polynomials. *Applicable Algebra in Engineering, Communication and Computing*, 7(5):351–364, 1996.

[11] G. Labahn M. Giesbrecht and W s. Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. *J. Symb. Comput.*, 44:943–959, 2009.

[12] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Trans. Inf. Theory it-15*, pages 122–127, 1969.

[13] Michael O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput*, 9:273–280, 1979.

[14] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press: Cambridge, New York, Port Melbourne, Madrid, Cape Town, second edition, 2003.

[15] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM '79*, pages 216–226. Springer-Verlag, 1979.

[16] Richard Zippel. Interpolating polynomials from their values. *J. Symb. Comput.*, 9(3):375–403, 1990.