Polynomial Interpolation

Michael Monagan

Department of Mathematics,
Simon Fraser University

Computers and Mathematics day, August 12, 2010

This is a joint work with Mahdi Javadi

Outline

- The black box model
- The sparse interpolation problem
- Previous work
- Our parallel algorithm
- Benchmarks and current work

The Black-Box model.

Let K is a ring e.g. $\mathbb{Z}, \mathbb{R}, GF(p)$.
Let f be a polynomial in $K[x_1, ..., x_n]$ given to us as a black-box.

- In this model all we may do is evaluate f at points in K^n.
- We call evaluations of f *probes* to the black-box.
- We want algorithms that minimize the number of probes.

Example of the black-box model $K = \mathbb{Q}$.

```
> A := Matrix([[x1,x2,x3],[x2,x1,x2],[x3,x2,x1]]);
```

$$A = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_1 & x_2 \\ x_3 & x_2 & x_1 \end{bmatrix}$$

```
> f := det(A);
```

$$f := x_1^3 - 2\,x_1 x_2^2 + 2\,x_3 x_2^2 - x_3^2 x_1$$

```
> factor(f);
```

$$(x_1 - x_3)\left(x_1^2 + x_1\,x_3 - 2\,x_2^2\right)$$

```
f := proc(x1::rational, x2::rational, x3::rational) :: rational;
   local A; A := Array(1..3,1..3);
   A[1,1] := x1; A[1,2] := x2; A[1,3] := x3;
   A[2,1] := x2; A[2,2] := x1; A[2,3] := x2;
   A[3,1] := x3; A[3,2] := x2; A[3,3] := x1;
   LinearAlgebra[Determinant](A);
end:
```

The Sparse Interpolation Problem.

Let $f = \sum_{i=1}^{t} c_i M_i$ where $c_i \in K$ and M_i are monomials in $x_1, ..., x_n$.
Assume we are given $d \geq \deg f$, and a term bound $T \geq t$.
In the black-box model, can we

1. Test if $f = 0$?
2. Determine c_i and M_i ?
3. Determine the factors of f ?

The Sparse Interpolation Problem.

Let $f = \sum_{i=1}^{t} c_i M_i$ where $c_i \in K$ and M_i are monomials in $x_1, ..., x_n$.
Assume we are given $d \geq \deg f$, and a term bound $T \geq t$.
In the black-box model, can we

1. Test if $f = 0$?
2. Determine c_i and M_i ?
3. Determine the factors of f ?

Yes, interpolate f using e.g. Newton interpolation.
If $|K| > d$ we can interpolate f with $(d+1)^n$ probes.

What if $f = 1 + x_1^d + x_2^d + ... + x_n^d$.
This polynomial is sparse – it has only $t = n + 1$ terms.

Sparse Interpolation Problem
Can we interpolate f in polynomial time in n, d, T ?

Previous work.

1978 Schwartz' zero test.

1979 Zippel's probabilistic sparse interpolation.

1988 Ben-Or/Tiwari's deterministic sparse interpolation.

1999 Huang and Rao's parallel algorithm.

2000 Kaltofen, Lee and Lobo's racing algorithm.

2006 Giesbrecht, Labahn and Lee's numerical method.

Testing if $f = 0$.

The Schwartz Lemma (Jack Schwartz, 1979)

Let $f \in K[x_1, ..., x_n]$ and $d \geq \deg f$.
Pick $\alpha_1, ..., \alpha_n$ from $S \subset K$ at random.
If $f \neq 0$ then

$$\mathrm{Prob}(f(\alpha_1, ..., \alpha_n) = 0) \ \leq \ \frac{d}{|S|}.$$

Example: Consider a prime $p > 2^{30}$ with $S = K = \mathbb{Z}_p$.
If $f(\alpha_1, ..., \alpha_n) = 0$ then

$$\mathrm{Prob}(f = 0) \ \geq \ 1 - \frac{d}{2^{30}}.$$

Zippel's probabilistic algorithm (1979).

Suppose p is a prime, $f \in \mathbb{Z}_p[x, y, z]$ and we know $\deg_x(f), \deg_y(f), \deg_z(f) \leq 15$.

Pick $\alpha \in \mathbb{Z}_p$ at random and interpolate, recursively,

$$f(x, y, \alpha) = \cdot x^9 y + \cdot x^5 y^4 + \cdot x^5 y^9$$

To interpolate z using Newton we need $\deg_z(f) = 15$ more bivariate images each of which requires $16 \times 16 = 256$ points.

Zippel's observation: If p is large and α is chosen at random, then

$$f(x, y, z) = A(z) x^9 y + B(z) x^5 y^4 + C(z) x^5 y^9 \quad w.h.p.$$

Zippel's probabilistic algorithm (1979).

Suppose p is a prime, $f \in \mathbb{Z}_p[x,y,z]$ and we know $\deg_x(f), \deg_y(f), \deg_z(f) \leq 15$.

Pick $\alpha \in \mathbb{Z}_p$ at random and interpolate, recursively,

$$f(x,y,\alpha) = \cdot\, x^9 y + \cdot\, x^5 y^4 + \cdot\, x^5 y^9$$

To interpolate z using Newton we need $\deg_z(f) = 15$ more bivariate images each of which requires $16 \times 16 = 256$ points.

Zippel's observation: If p is large and α is chosen at random, then

$$f(x,y,z) = A(z)x^9 y + B(z)x^5 y^4 + C(z)x^5 y^9 \quad w.h.p.$$

Zippel's idea: Get the next bivariate image for $f(x,y,\beta)$ by picking $\beta, a_1, b_1, a_2, b_2, a_3, b_3$ at random and solving, for A, B, C,

$$\begin{aligned} f(a_1,b_1,\beta) &= A a_1^9 b_1 + B a_1^5 b_1^4 + C b_1 \\ f(a_2,b_2,\beta) &= A a_2^9 b_2 + B a_2^5 b_2^4 + C b_2 \\ f(a_3,b_3,\beta) &= A a_3^9 b_3 + B a_3^5 b_3^4 + C b_3 \end{aligned}$$

This linear system is non-singular w.h.p. $\implies$ 3 probes instead of 256.

Zippel's probabilistic algorithm (1979).

Suppose p is a prime, $f \in \mathbb{Z}_p[x, y, z]$ and we know $\deg_x(f), \deg_y(f), \deg_z(f) \leq 15$.

Pick $\alpha \in \mathbb{Z}_p$ at random and interpolate, recursively,

$$f(x, y, \alpha) = \cdot\, x^9 y + \cdot\, x^5 y^4 + \cdot\, x^5 y^9$$

To interpolate z using Newton we need $\deg_z(f) = 15$ more bivariate images each of which requires $16 \times 16 = 256$ points.

Zippel's observation: If p is large and α is chosen at random, then

$$f(x, y, z) = A(z)x^9 y + B(z)x^5 y^4 + C(z)x^5 y^9 \quad w.h.p.$$

Zippel's idea: Get the next bivariate image for $f(x, y, \beta)$ by picking $\beta, a_1, b_1, a_2, b_2, a_3, b_3$ at random and solving, for A, B, C,

$$\begin{aligned} f(a_1, b_1, \beta) &= A a_1^9 b_1 + B a_1^5 b_1^4 + C b_1 \\ f(a_2, b_2, \beta) &= A a_2^9 b_2 + B a_2^5 b_2^4 + C b_2 \\ f(a_3, b_3, \beta) &= A a_3^9 b_3 + B a_3^5 b_3^4 + C b_3 \end{aligned}$$

This linear system is non-singular w.h.p. $\implies$ 3 probes instead of 256.

- Zippel's algorithm is probabilistic and does $O(ndt)$ probes.

Ben-Or and Tiwari's algorithm (1988).

Let $f = \sum_{i=1}^{t} c_i M_i$ where $c_i \in \mathbb{Z}$ and $M_i = x_1^{d_{i1}} x_2^{d_{i2}} \cdots x_n^{d_{in}}$.

Input $T \geq t$.

Step 1 For $i = 0 \dots 2T-1$ compute $v_i = f(2^i, 3^i, 5^i, \dots, p_n^i)$.

Step 2 Compute the linear generator $\Lambda(z)$ for the sequence $v_0, v_1, \dots, v_{2T-1}$ using the Berlekamp/Massey algorithm.

Theorem: $\Lambda(z) = \prod_{i=1}^{t} (z - M_i(2, 3, 5, \dots, p_n))$.

Step 3 Compute the integer roots of $\Lambda(z)$: $m_1, \dots, m_t$.

Step 4 Divide m_i by p_j to determine $deg_{x_j}(M_i)$ hence M_i.

Step 5 Solve (a transposed Vandermode system) for the coefficients c_i.

The Ben-Or/Tiwari algorithm contd.

- Ben-Or/Tiwari is deterministic and does $2T$ probes.
- But the integers $f(2^i, 3^i, 5^i, ..., p_n^i)$ are as large as p_n^{2Td}, which can be very big. E.g. if $n = 10, d = 50, t = 100$, $p_n^{2Td} > 14,000$ digits!
- Worse, Kaltofen and Lobo observed that rational numbers in the Berlekamp-Massey algorithm get $t = 100$ times larger still !!

The Ben-Or/Tiwari algorithm contd.

- Ben-Or/Tiwari is deterministic and does $2T$ probes.
- But the integers $f(2^i, 3^i, 5^i, ..., p_n^i)$ are as large as p_n^{2Td}, which can be very big. E.g. if $n = 10, d = 50, t = 100$, $p_n^{2Td} > 14,000$ digits!
- Worse, Kaltofen and Lobo observed that rational numbers in the Berlekamp-Massey algorithm get $t = 100$ times larger still !!

Solution: Run Ben-Or/Tiwari modulo a prime p satisfying

$$p > \max_i M_i(2, 3, 5, ..., p_n) < p_n^{d}.$$

Still $n = 10, d = 50 \Longrightarrow p > 10^{74}$.

Huang and Rao's algorithm for $K = GF(q)$ (1999).

Idea: Replace the primes $2, 3, 5, \ldots$ in Ben-Or/Tiwari by irreducible polynomials $y - a_1, y - a_2, \ldots$ for $a_j \in GF(q)$.

How do we evaluate the back box at polyomials
$f\left((y-a_1)^i, (y-a_2)^i, \ldots, (y-a_n)^i\right)$, for $i = 0, 1, \ldots, 2T-1$?

Huang and Rao's algorithm for $K = GF(q)$ (1999).

Idea: Replace the primes $2, 3, 5, \ldots$ in Ben-Or/Tiwari by irreducible polynomials $y - a_1, y - a_2, \ldots$ for $a_j \in GF(q)$.

How do we evaluate the back box at polyomials $f\left((y-a_1)^i, (y-a_2)^i, \ldots, (y-a_n)^i\right)$, for $i = 0, 1, \ldots, 2T-1$?

Solution: interpolate $f((y-a_1)^i, (y-a_2)^i, \ldots, (y-a_n)^i) \in GF(q)[y]$ from $di+1$ values for y in $GF(q)$.

- Requires $q > 8d^2t^2$.
- Does $O(dt^2)$ probes.
- Needs to factor $\Lambda(x, y) \in GF(q)[x, y]$.

Kaltofen, Lee and Lobo's algorithm for GF(q).

In 2000 Kaltofen, Lee and Lobo presented a hybrid of Zippel's algorithm and the Ben-Or/Tiwari algorithm.

Their algorithm modifies Zippel's algorithm. Consider

$$f(x,y,z) = 7z^2x^7y^3 + (3z^5+5)xy^4 + 7z^{11}x$$

- For univariate interpolation, they race Newton's interpolation with univariate Ben-Or/Tiwari using same evaluation points.
- This reduces the number of probes from $O(ndt)$ to $O(nt)$.
- But this sequentializes the algorithm!

Comparison Chart

For applications where we can choose the prime p:

Alg.	# Probes	Deterministic?	Parallel?	Prime
Ben-Or/Tiwari 1988	$O(t)$	Las Vegas	Yes	$p > p_n^d$
Huang/Rao 1990	$O(dt^2)$	Las Vegas	Yes	$p > 8d^2t^2$
Zippel 1979	$O(ndt)$	Monte-Carlo	Some	$p \gg nt$
Kaltofen et. al. 2000	$O(nt)$	Monte-Carlo	Less	$p \gg nt$
Javadi/Monagan 2010	$O(nt)$	Monte-Carlo	Yes!	$p \gg (n+d)t^2$

Three problems:

- Medium: $n = 10, d = 20, t = 10^2$.
- Big: $n = 15, d = 40, t = 10^4$.
- Very Big: $n = 20, d = 100, t = 10^6$.

Alg.	Prime	Medium	Big	Very Big
Ben-Or/Tiwari	$p > p_n^d$	2^{96}	2^{223}	2^{615}
Huang/Rao	$p > 8d^2t^2$	2^{25}	2^{41}	2^{56}
Zippel	$p \gg nt$	2^{10}	2^{17}	2^{24}
Kaltofen et. al.	$p \gg nt$	2^{10}	2^{17}	2^{24}
Javadi/Monagan	$p \gg (n+d)t^2$	2^{18}	2^{32}	2^{47}

Our New Algorithm: The Idea

1. Choose non-zero $\alpha_1, \ldots, \alpha_n$ at random from $\mathbb{Z}_p$.
2. Evaluate $f(\alpha_1^i, \ldots, \alpha_j^i, \ldots, \alpha_n^i)$ for $i = 0 \ldots 2T - 1$ and compute $\Lambda_0(z) \in \mathbb{Z}_p[z]$.
3. Find the roots of $\Lambda_0(z) : r_1, \ldots, r_t$ using Rabin's algorithm.
   We have $\{r_1, \ldots, r_t\} = \{m_1, \ldots, m_t\}$ where $m_i = M_i(\alpha_1, \ldots, \alpha_n)$.
4. To determine the monomials $M_i(x_1, ..., x_n) = x_1^{d_{i1}} \cdots x_n^{d_{in}}$:

Our New Algorithm: The Idea

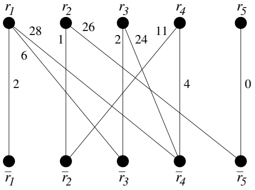
1. Choose non-zero $\alpha_1, \ldots, \alpha_n$ at random from $\mathbb{Z}_p$.
2. Evaluate $f(\alpha_1^i, \ldots, \alpha_{\mathbf{j}}^{\mathbf{i}}, \ldots, \alpha_n^i)$ for $i = 0 \ldots 2T - 1$ and compute $\Lambda_0(z) \in \mathbb{Z}_p[z]$.
3. Find the roots of $\Lambda_0(z)$: $r_1, \ldots, r_t$ using Rabin's algorithm.
   We have $\{r_1, \ldots, r_t\} = \{m_1, \ldots, m_t\}$ where $m_i = M_i(\alpha_1, \ldots, \alpha_n)$.
4. To determine the monomials $M_i(x_1, ..., x_n) = x_1^{d_{i1}} \cdots x_n^{d_{in}}$:
   For each x_j do the following in parallel:
   4.1 Choose $\beta_j \neq \alpha_j$ at random from $\mathbb{Z}_p$.
   4.2 Evaluate $f(\alpha_1^i, \ldots, \beta_{\mathbf{j}}^{\mathbf{i}}, \ldots, \alpha_n^i)$ for $i = 0 \ldots 2t - 1$ and compute $\Lambda_j(z)$.
   Let $\bar{r}_1, \ldots, \bar{r}_t$ denote the roots of $\Lambda_j(z)$ and $\bar{m}_i = M_i(\alpha_1, \ldots, \beta_j, \ldots, \alpha_n)$.
   We have $\{\bar{r}_1, \ldots, \bar{r}_t\} = \{\bar{m}_1, \ldots, \bar{m}_t\}$. Observe:

$$\frac{\bar{m}_i}{m_i} = \left(\frac{\beta_j}{\alpha_j}\right)^{d_{ij}} \Rightarrow \bar{m}_i = \left(\frac{\beta_j}{\alpha_j}\right)^{d_{ij}} m_i \Rightarrow \Lambda_j\left(\left(\frac{\beta_j}{\alpha_j}\right)^{d_{ij}} m_i\right) = 0.$$

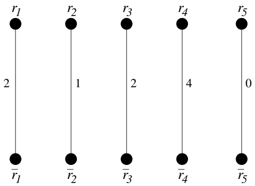
   4.3 For $i = 1 \ldots t$ do
      4.3.1 For $s = 0 \ldots d$ do if $\Lambda_j\left(\left(\frac{\beta_j}{\alpha_j}\right)^{s} r_i\right) = 0$ then $d_{ij} = s$ w.h.p.

Our New Algorithm (contd.)

We construct the following bipartite graph. r_i is connected to $\bar{r}_j$ with the weight e iff $\bar{r}_j = r_i(\frac{\beta_j}{\alpha_j})^e$.



This graph has a unique perfect matching



which tells us the degree of all monomials in x_j.

Our New Algorithm.

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \dots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

Choose $\alpha_1, \dots, \alpha_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random.
Choose $\beta_1, \dots, \beta_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random s.t. $\mathrm{order}(\beta_k/\alpha_k) > d$.
for k from 0 to n in **parallel do**
    k=0: Compute $\Lambda_0(x)$ from $f(\alpha_1^i, \dots, {\alpha_k}^i, \dots \alpha_n^i)$ for $0 \leq i \leq 2T - 1$.
    k>0: Compute $\Lambda_{k+1}(x)$ from $f(\alpha_1^i, \dots, {\beta_k}^i, \dots, \alpha_n^i)$ for $0 \leq i \leq 2T - 1$.
end for
Set $t = \max_{i=1}^{n+1} \deg \Lambda_i(z)$. If $\deg(\Lambda_i) < t$ **return** FAIL.
Compute $\{r_1, \dots, r_t\}$ the set of distinct roots of $\Lambda_1(z)$.
for k from 1 to n in **parallel do**
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching **return** FAIL
    else we have determined $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$.
end for
Solve for the unknown coefficients c_i and let $g = \sum_{i=1}^{t} c_i M_i$.
Check if $g = f$: choose $a_1, \dots, a_n$ from $\mathbb{Z}_p$ at random.
If $g(a_1, \dots, a_n) = f(a_1, \dots, a_n)$ **return** g **else return** FAIL.

Our New Algorithm

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

Choose $\alpha_1, \ldots, \alpha_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random.
Choose $\beta_1, \ldots, \beta_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random s.t. $\mathrm{order}(\beta_k/\alpha_k) > d$.
for k from 0 to n in **parallel do**
    k=0: Compute $\Lambda_0(x)$ from $f(\alpha_1^i, \ldots, {\alpha_k}^i, \ldots \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
    k>0: Compute $\Lambda_{k+1}(x)$ from $f(\alpha_1^i, \ldots, {\beta_k}^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
end for
Set $t = \max_{i=1}^{n+1} \deg \Lambda_i(z)$. If $\deg(\Lambda_i) < t$ **return** FAIL.
Compute $\{r_1, \ldots, r_t\}$ the set of distinct roots of $\Lambda_1(z)$.
for k from 1 to n in **parallel do**
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching **return** FAIL
    else we have determined $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$.
end for
Solve for the unknown coefficients c_i and let $g = \sum_{i=1}^{t} c_i M_i$.
Check if $g = f$: choose $a_1, \ldots, a_n$ from $\mathbb{Z}_p$ at random.
If $g(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$ **return** g **else return** FAIL.

Our New Algorithm

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

Choose $\alpha_1, \ldots, \alpha_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random.
Choose $\beta_1, \ldots, \beta_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random s.t. $\mathrm{order}(\beta_k / \alpha_k) > d$.
for k from 0 to n in **parallel do**
    k=0: Compute $\Lambda_0(x)$ from $f(\alpha_1^i, \ldots, {\alpha_k}^i, \ldots \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
    k>0: Compute $\Lambda_{k+1}(x)$ from $f(\alpha_1^i, \ldots, {\beta_k}^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
end for
Set $t = \max_{i=1}^{n+1} \deg \Lambda_i(z)$. If $\deg(\Lambda_i) < t$ **return** FAIL.
Compute $\{r_1, \ldots, r_t\}$ the set of distinct roots of $\Lambda_1(z)$.
for k from 1 to n in **parallel do**
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching **return** FAIL
    else we have determined $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$.
end for
Solve for the unknown coefficients c_i and let $g = \sum_{i=1}^{t} c_i M_i$.
Check if $g = f$: choose $a_1, \ldots, a_n$ from $\mathbb{Z}_p$ at random.
If $g(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$ **return** g **else return** FAIL.

Our New Algorithm

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

Choose $\alpha_1, \ldots, \alpha_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random.
Choose $\beta_1, \ldots, \beta_n$ from $\mathbb{Z}_p \backslash \{0\}$ at random s.t. $\mathrm{order}(\beta_k/\alpha_k) > d$.
for k from 0 to n in **parallel do**
    k=0: Compute $\Lambda_0(x)$ from $f(\alpha_1^i, \ldots, \alpha_k{}^i, \ldots \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
    k>0: Compute $\Lambda_{k+1}(x)$ from $f(\alpha_1^i, \ldots, \beta_k{}^i, \ldots, \alpha_n^i)$ for $0 \leq i \leq 2T-1$.
end for
Set $t = \max_{i=1}^{n+1} \deg \Lambda_i(z)$. If $\deg(\Lambda_i) < t$ **return** FAIL.
Compute $\{r_1, \ldots, r_t\}$ the set of distinct roots of $\Lambda_1(z)$.
for k from 1 to n in **parallel do**
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching **return** FAIL
    else we have determined $\deg_{x_k}(M_i)$ for $1 \leq i \leq t$.
end for
Solve for the unknown coefficients c_i and let $g = \sum_{i=1}^{t} c_i M_i$.
Check if $g = f$: choose $a_1, \ldots, a_n$ from $\mathbb{Z}_p$ at random.
If $g(a_1, \ldots, a_n) = f(a_1, \ldots, a_n)$ **return** g **else return** FAIL.

Our New Algorithm

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

```
Choose α_1, ..., α_n from Z_p\{0} at random.
Choose β_1, ..., β_n from Z_p\{0} at random s.t. order(β_k/α_k) > d.
for k from 0 to n in parallel do
    k=0: Compute Λ_0(x) from f(α_1^i, ..., α_k^i, ... α_n^i) for 0 ≤ i ≤ 2T − 1.
    k>0: Compute Λ_{k+1}(x) from f(α_1^i, ..., β_k^i, ..., α_n^i) for 0 ≤ i ≤ 2T − 1.
end for
Set t = max_{i=1}^{n+1} deg Λ_i(z). If deg(Λ_i) < t return FAIL.
Compute {r_1, ..., r_t} the set of distinct roots of Λ_1(z).
for k from 1 to n in parallel do
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching return FAIL
    else we have determined deg_{x_k}(M_i) for 1 ≤ i ≤ t.
end for
Solve for the unknown coefficients c_i and let g = Σ_{i=1}^t c_i M_i .
Check if g = f: choose a_1, ..., a_n from Z_p at random.
If g(a_1, ..., a_n) = f(a_1, ..., a_n) return g else return FAIL.
```

Our New Algorithm

Require: A polynomial $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ input as a black box.
Require: A degree bound $d \geq \deg(f)$ and a term bound $T \geq t$.

```
Choose α_1, ..., α_n from Z_p\{0} at random.
Choose β_1, ..., β_n from Z_p\{0} at random s.t. order(β_k/α_k) > d.
for k from 0 to n in parallel do
    k=0: Compute Λ_0(x) from f(α_1^i, ..., α_k^i, ... α_n^i) for 0 ≤ i ≤ 2T − 1.
    k>0: Compute Λ_{k+1}(x) from f(α_1^i, ..., β_k^i, ..., α_n^i) for 0 ≤ i ≤ 2T − 1.
end for
Set t = max_{i=1}^{n+1} deg Λ_i(z). If deg(Λ_i) < t return FAIL.
Compute {r_1, ..., r_t} the set of distinct roots of Λ_1(z).
for k from 1 to n in parallel do
    Construct the bi-partite graph G_k as just described.
    If G_k does not have a unique perfect matching return FAIL
    else we have determined deg_{x_k}(M_i) for 1 ≤ i ≤ t.
end for
Solve for the unknown coefficients c_i and let g = Σ_{i=1}^t c_i M_i .
Check if g = f: choose a_1, ..., a_n from Z_p at random.
If g(a_1, ..., a_n) = f(a_1, ..., a_n) return g else return FAIL.
```

Algorithm failure probability

Theorem 1: For random non-zero $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$, the probability that two or more monomials M_i evaluate to the same value is $\leq \binom{t}{2} \frac{d}{p-1}$.

Theorem 2: If $\deg(\Lambda_0) = \deg(\Lambda_j) = t$, then the probability that we will not be able to uniquely compute the degrees in x_j is at most $\frac{d^2 t^2}{4\phi(p-1)}$.

Algorithm failure probability

Theorem 1: For random non-zero $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$, the probability that two or more monomials M_i evaluate to the same value is $\leq \binom{t}{2} \frac{d}{p-1}$.

Theorem 2: If $\deg(\Lambda_0) = \deg(\Lambda_j) = t$, then the probability that we will not be able to uniquely compute the degrees in x_j is at most $\frac{d^2 t^2}{4\phi(p-1)}$.

Proof of Theorem 1: Consider

$$A = \prod_{1 \leq i < j \leq t} \left(M_i(x_1, \ldots, x_n) - M_j(x_1, \ldots, x_n) \right).$$

Observe that $A(\alpha_1, \ldots, \alpha_n) = 0$ iff two monomial evaluations collide.

Applying the Schwartz lemma, since $\deg(M_i) \leq d$ we have

$$\mathrm{Prob}(A(\alpha_1, \ldots, \alpha_n) = 0) \leq \frac{\deg A}{|S|} \leq \frac{\binom{t}{2} d}{p-1}.$$

Optimizations

Theorem 3 : The algorithm makes $2(n+1)T$ probes, does $O((n+1)t^2 + \log(p)t^2 + ndt^2)$ other work, and succeeds with probability at least $1 - \frac{(n+1)d^2t^2}{2\phi(p-1)}$.

Optimizations

Theorem 3 : The algorithm makes $2(n+1)T$ probes, does $O((n+1)t^2 + \log(p)t^2 + ndt^2)$ other work, and succeeds with probability at least $1 - \frac{(n+1)d^2t^2}{2\phi(p-1)}$.

- Pick p s.t. $p = 2q + 1$ with q prime to maximize $\phi(p-1)$.
- To compute the degrees of the monomials in the last variable x_n, we do not need to do any more probes to the black box. We have

$$m_i = \alpha_1^{d_{i1}} \times \cdots \times \alpha_{n-1}^{d_{i(n-1)}} \times \alpha_n^{d_{in}}.$$

- When determining the degree of M_i in x_j, stop when the first s gives a root – it's the right degree w.h.p.

Optimizations

Theorem 3 : The algorithm makes $2(n+1)T$ probes, does $O((n+1)t^2 + \log(p)t^2 + ndt^2)$ other work, and succeeds with probability at least $1 - \frac{(n+1)d^2t^2}{2\phi(p-1)}$.

- Pick p s.t. $p = 2q+1$ with q prime to maximize $\phi(p-1)$.
- To compute the degrees of the monomials in the last variable x_n, we do not need to do any more probes to the black box. We have

$$m_i = \alpha_1^{d_{i1}} \times \cdots \times \alpha_{n-1}^{d_{i(n-1)}} \times \alpha_n^{d_{in}}.$$

- When determining the degree of M_i in x_j, stop when the first s gives a root – it's the right degree w.h.p.

Theorem 3 : The algorithm makes $2nT$ probes, does $O(nt^2 + \log(p)t^2 + dt^2)$ other work, and succeeds with probability at least $1 - \frac{(n+d^2)t^2}{p-1}$.

Benchmarks

Random polynomials in $n = 12$ variables with approximately $t = 2^i$ terms of total degree 30 using $T = t$ and $d = 30$.

i	t	New Algorithm		Zippel		ProtoBox
		Time (4 cores)	Probes	Time	Probes	Probes
4	15	0.00 (0.00)	360	0.20	10230	470
5	32	0.02 (0.01)	768	0.54	18879	962
6	63	0.04 (0.02)	1512	1.79	36735	1856
7	127	0.15 (0.05)	3048	6.10	69595	3647
8	255	0.54 (0.17)	6120	22.17	134664	7055
9	507	2.01 (0.60)	12168	83.44	259594	13440
10	1019	7.87 (2.33)	24456	316.23	498945	26077
11	2041	31.0 (9.16)	48984	1195.13	952351	DNF
12	4074	122.3 (35.9)	97776	4575.83	1841795	DNF
13	8139	484.6 (141.)	195336	$>$10000	-	DNF

Timings are in CPU seconds on an Intel Corei7.
The parallel implementation was done in Cilk.

Current work

i	t	1 core				4 cores		
		time	roots	solve	probes	time 1	time 2	speedup
8	255	0.54	0.05	0.00	0.41	0.18	0.17	(3x)
9	507	2.02	0.18	0.02	1.48	0.67	0.60	(3.02x)
10	1019	7.94	0.65	0.08	5.76	2.58	2.33	(3.08x)
11	2041	31.3	2.47	0.32	22.7	9.94	9.16	(3.15x)
12	4074	122.3	9.24	1.26	90.0	38.9	35.9	(3.14x)
13	8139	484.6	34.7	5.02	357.3	152.5	141.5	(3.17x)

Amdahl's law: Speedup $\leq \dfrac{Tot}{\frac{Tot-Seq}{\#cores} + Seq}$.

For $i = 13$ this gives 3.21 for 4 cores and 6.31 for 12 cores.

Current work

i	t	1 core				4 cores		
		time	roots	solve	probes	time 1	time 2	speedup
8	255	0.54	0.05	0.00	0.41	0.18	0.17	(3x)
9	507	2.02	0.18	0.02	1.48	0.67	0.60	(3.02x)
10	1019	7.94	0.65	0.08	5.76	2.58	2.33	(3.08x)
11	2041	31.3	2.47	0.32	22.7	9.94	9.16	(3.15x)
12	4074	122.3	9.24	1.26	90.0	38.9	35.9	(3.14x)
13	8139	484.6	34.7	5.02	357.3	152.5	141.5	(3.17x)

Amdahl's law: Speedup $\leq \dfrac{Tot}{\frac{Tot - Seq}{\#cores} + Seq}$.

For $i = 13$ this gives 3.21 for 4 cores and 6.31 for 12 cores.

We are currently implementing fast arithmetic in $\mathbb{Z}_p[x]$ for 31 and 63 bit primes to speed up the $O(t^2 \log(p))$ root finding step which is the sequential bottleneck, and also to handle large values of t.

For $i = 13$ this would give 3.89 for 4 cores and 9.95 for 12 cores.

Giesbrecht, Labahn and Lee's numerical method.

A modification of Ben-Or/Tiwari for polynomials with numerical coefficients.
Idea: evaluate at powers of primitive elements in $\mathbb{C}$ of relatively prime order.

Pick $w_1, \ldots, w_n$ of order $q_1, \ldots, q_n$ s.t. $q_j > d$, $\gcd(q_j, q_k) = 1$.

Evaluate $f(w_1^i, \ldots, w_n^i)$, for $i = 0, 1, ..., 2T - 1$ and compute the roots $m_1, ..., m_t$ of $\Lambda(x)$ numerically. We have

$$m_i = M_i(w_1, \ldots, w_n) = w_1^{d_{i1}} \times w_2^{d_{i2}} \times \cdots \times w_n^{d_{in}} = w^{\frac{p-1}{q_1} d_{i1} + \cdots + \frac{p-1}{q_n} d_{in}}$$

where w has order $q_1 \cdot q_2 \cdots q_n$. Now take logarithms to the base w:

$$\Rightarrow \quad \log_w m_i = \frac{p-1}{q_1} d_{i1} + \cdots + \frac{p-1}{q_j} d_{ij} + \cdots + \frac{p-1}{q_n} d_{in}$$

Round $\log_w(m_i)$ and solve this modulo q_j to get $d_{i,j}$.

Giesbrecht, Labahn and Lee's numerical method.

A modification of Ben-Or/Tiwari for polynomials with numerical coefficients.
Idea: evaluate at powers of primitive elements in $\mathbb{C}$ of relatively prime order.

Pick $w_1, \ldots, w_n$ of order $q_1, \ldots, q_n$ s.t. $q_j > d$, $\gcd(q_j, q_k) = 1$.

Evaluate $f(w_1^i, \ldots, w_n^i)$, for $i = 0, 1, ..., 2T - 1$ and compute the roots $m_1, ..., m_t$ of $\Lambda(x)$ numerically. We have

$$m_i = M_i(w_1, \ldots, w_n) = w_1^{d_{i1}} \times w_2^{d_{i2}} \times \cdots \times w_n^{d_{in}} = w^{\frac{p-1}{q_1} d_{i1} + \cdots + \frac{p-1}{q_n} d_{in}}$$

where w has order $q_1 \cdot q_2 \cdots q_n$. Now take logarithms to the base w:

$$\Rightarrow \quad \log_w m_i = \frac{p-1}{q_1} d_{i1} + \cdots + \frac{p-1}{q_j} d_{ij} + \cdots + \frac{p-1}{q_n} d_{in}$$

Round $\log_w(m_i)$ and solve this modulo q_j to get $d_{i,j}$.

For applications where we can pick p, this can work in $\mathbb{Z}_p$ as follows:

- Pick $p = q_1 \cdot q_2 \cdots q_n + 1$ s.t. $q_i > d$ and $\gcd(q_i, q_j) = 1$ until p is prime.
- The discrete log is efficient if we choose $p - 1$ with no large prime factors.
- $O(T)$ probes but requires $p > (d+1)^n$ which may be big.

Thank you.