# On Lacunary Polynomial Perfect Powers

Mark Giesbrecht Daniel S. Roche

ORCCA
Ontario Research Centre for Computer Algebra

Symbolic Computation Group
School of Computer Science
University of Waterloo

UNIVERSITY OF Waterloo

ISSAC 2008
RISC Linz
Hagenberg, Austria
23 July 2008

# Polynomial Representations

A polynomial which is easy to write:

$$f(x) = x^{123705}y^{9432} + 9\,x^{95911}y^{7510} + 3\,x^{82470}y^{6288} + 27\,x^{68117}y^{5588} + 18\,x^{54676}y^{4366}$$
$$+ 3\,x^{41235}y^{3144} + 27\,x^{40323}y^{3666} + 27\,x^{26882}y^{2444} + 9\,x^{13441}y^{1222} + 1$$

How to represent this polynomial?

# Polynomial Representations

A polynomial which is easy to write:

$$f(x) = x^{123705}y^{9432} + 9\,x^{95911}y^{7510} + 3\,x^{82470}y^{6288} + 27\,x^{68117}y^{5588} + 18\,x^{54676}y^{4366}$$
$$+ 3\,x^{41235}y^{3144} + 27\,x^{40323}y^{3666} + 27\,x^{26882}y^{2444} + 9\,x^{13441}y^{1222} + 1$$

## Dense Representation

For a degree-$n$ polynomial in $k$ variables:

- Store a $k$-dimensional array of every possible coefficient
- Size $\approx n^k$
- This example: more than 1 GB!

# Polynomial Representations

A polynomial which is easy to write:

$$f(x) = x^{123705}y^{9432} + 9\,x^{95911}y^{7510} + 3\,x^{82470}y^{6288} + 27\,x^{68117}y^{5588} + 18\,x^{54676}y^{4366}$$
$$+ 3\,x^{41235}y^{3144} + 27\,x^{40323}y^{3666} + 27\,x^{26882}y^{2444} + 9\,x^{13441}y^{1222} + 1$$

## Sparse Representation

For a degree-$n$ polynomial in $k$ variables with $t$ terms:

- Store a list of $t$ coefficient-exponent tuples
- Size $\approx kt \log_2 n$
- This example: less than 1 KB

# Polynomial Representations

A polynomial which is easy to write:

$$f(x) = x^{123705}y^{9432} + 9\,x^{95911}y^{7510} + 3\,x^{82470}y^{6288} + 27\,x^{68117}y^{5588} + 18\,x^{54676}y^{4366}$$
$$+ 3\,x^{41235}y^{3144} + 27\,x^{40323}y^{3666} + 27\,x^{26882}y^{2444} + 9\,x^{13441}y^{1222} + 1$$

## Sparse Representation

For a degree-$n$ polynomial in $k$ variables with $t$ terms:

- Store a list of $t$ coefficient-exponent tuples
- Size $\approx kt \log_2 n$
- This example: less than 1 KB
- More natural representation
- Default in most CAS
- Can be exponentially more compact

# Computing with Polynomials

Lots of things are easy when polynomials are dense:

- GCD, factorization, Euclidean division
- Relative Primality
- Square-freeness
- Divisibility
- Reducibility
- Evaluation, Differentiation
- Interpolation
- Root finding
- Low-degree factors
- Jacobi symbols, perfect square detection

# Computing with Polynomials

Some things are hard when polynomials are sparse:

- GCD, factorization, Euclidean division
- Relative Primality (Plaisted '84)
- Square-freeness (Karpinski & Shparlinski '99)
- Divisibility
- Reducibility
- Evaluation, Differentiation
- Interpolation
- Root finding
- Low-degree factors
- Jacobi symbols, perfect square detection

# Computing with Polynomials

Some things might be hard when polynomials are sparse:

- GCD, factorization, Euclidean division
- Relative Primality (Plaisted '84)
- Square-freeness (Karpinski & Shparlinski '99)
- Divisibility
- Reducibility
- Evaluation, Differentiation
- Interpolation
- Root finding
- Low-degree factors
- Jacobi symbols, perfect square detection

# Computing with Polynomials

Some things are easy when polynomials are sparse:

- GCD, factorization, Euclidean division
- Relative Primality (Plaisted '84)
- Square-freeness (Karpinski & Shparlinski '99)
- Divisibility
- Reducibility
- Evaluation, Differentiation
- Sparse Interpolation
  (Ben-Or & Tiwari '88; Kaltofen & Lee '03; G. & R. '07)
- Root finding (Cucker, Koiran, Smale '99)
- Low-degree factors (Lenstra '99; Kaltofen & Koiran '05,'06)
- Jacobi symbols, perfect square detection (Shparlinski '00)

# Computing with Polynomials

Some things are easy when polynomials are sparse:

- GCD, factorization, Euclidean division
- Relative Primality (Plaisted '84)
- Square-freeness (Karpinski & Shparlinski '99)
- Divisibility
- Reducibility
- Evaluation, Differentiation
- Sparse Interpolation
  (Ben-Or & Tiwari '88; Kaltofen & Lee '03; G. & R. '07)
- Root finding (Cucker, Koiran, Smale '99)
- Low-degree factors (Lenstra '99; Kaltofen & Koiran '05,'06)
- Jacobi symbols, perfect square detection (Shparlinski '00)
- Perfect power detection Today!

# Back to the example

$$f(x) = x^{123705}y^{9432} + 9\,x^{95911}y^{7510} + 3\,x^{82470}y^{6288} + 27\,x^{68117}y^{5588} + 18\,x^{54676}y^{4366}$$
$$+ 3\,x^{41235}y^{3144} + 27\,x^{40323}y^{3666} + 27\,x^{26882}y^{2444} + 9\,x^{13441}y^{1222} + 1$$
$$= \left(x^{41235}y^{3144} + 3\,x^{13441}y^{1222} + 1\right)^3$$
$$r = 3$$
$$h(x) = x^{41235}y^{3144} + 3\,x^{13441}y^{1222} + 1$$

- We will use this notation consistently.

# Problems to Solve

1. Given $f$, determine whether $f = h^r$ for any $h$ and $r \geq 2$.
2. If so, find one such $r$.
3. Given $r$, find one such $h$.

Algorithm Requirements:

- Cost polynomial in $k$, $t$, and $\log n$
- When R $= \mathbb{Z}$, also polynomial in $\log \|f\|_\infty$

# Problems to Solve

1. Given $f$, determine whether $f = h^r$ for any $h$ and $r \geq 2$.
2. If so, find one such $r$.
3. Given $r$, find one such $h$.

Algorithm Requirements:

- Cost polynomial in $k$, $t$, and $\log n$
- When R $= \mathbb{Z}$, also polynomial in $\log \|f\|_\infty$

First, we solve (1) and (2) for univariate integer polynomials.

# Perfect Power Detection Algorithm

**Input:** $f \in \mathbb{Z}[x]$

**Output:** $r \geq 2$ s.t. $f = h^r$ for some $h$, or FALSE

**for** each possible $r$ **do**

Choose random $\alpha \in \mathbb{Z}$

**if** $f(\alpha)$ is a perfect $r^{\text{th}}$ power **then**

**return** $r$

**end do**

**return** FALSE

Problem: Can't evaluate over $\mathbb{Z}$

# Perfect Power Detection Algorithm

**Input:** $f \in \mathbb{Z}[x]$

**Output:** $r \geq 2$ s.t. $f = h^r$ for some $h$, or FALSE

**for** each possible $r$ **do**

Probabilistically choose $p$ with $p \nmid \operatorname{disc}(f)$

Choose random $\alpha \in \mathbb{F}_p$

**if** $f(\alpha)$ is a perfect $r^{\text{th}}$ power **then**

**return** $r$

**end do**

**return** FALSE

Problem: Can't determine $r^{\text{th}}$-poweredness over $\mathbb{F}_p$.

# Perfect Power Detection Algorithm

**Input:** $f \in \mathbb{Z}[x]$

**Output:** $r \geq 2$ s.t. $f = h^r$ for some $h$, or FALSE

**for** each possible prime $r$ **do**

Probabilistically choose $p$ with $p \nmid \mathrm{disc}(f)$

$q \leftarrow p^{r-1}$

Choose random $\alpha \in \mathbb{F}_q$

**if** $f(\alpha)^{(q-1)/r} = 1$ **then**

**return** $r$

**end do**

**return** FALSE

Problem: Need some deeper math

# Perfect Power Detection Algorithm

**Input:** $f \in \mathbb{Z}[x]$

**Output:** $r \geq 2$ s.t. $f = h^r$ for some $h$, or FALSE

**for** each prime $r < 2\log_2 \|f\|_1$ **do**

Probabilistically choose $p$ with $p \nmid \mathrm{disc}(f)$

$q \leftarrow p^{r-1}$

Choose random $\alpha \in \mathbb{F}_q$

**if** $f(\alpha)^{(q-1)/r} = 1$ **then**

**return** $r$

**end do**

**return** FALSE

Problem: Need some deeper math

1. Can restrict to $r < 2\log_2 \|f\|_1$

# Perfect Power Detection Algorithm

**Input:** $f \in \mathbb{Z}[x]$

**Output:** $r \geq 2$ s.t. $f = h^r$ for some $h$, or FALSE

**for** each prime $r < 2\log_2 \|f\|_1$ **do**

Probabilistically choose $p$ with $p \nmid \operatorname{disc}(f)$

$q \leftarrow p^{r-1}$

Choose random $\alpha_1, \alpha_2, \ldots, \alpha_5 \in \mathbb{F}_q$

**if** $f(\alpha_1)^{(q-1)/r} = \cdots = f(\alpha_5)^{(q-1)/r} = 1$ **then**

**return** $r$

**end do**

**return** FALSE

Problem: Need some deeper math

1. Can restrict to $r < 2\log_2 \|f\|_1$
2. Five evaluations guarantees high probability of success

# Bound on $r$

## Theorem

*If $f \in \mathbb{Z}[x]$ is a perfect $r^{\text{th}}$ power and has at least 2 terms, then*

$$r \leq 2 \log_2 \|f\|_1.$$

- Proof arises from orthogonality of DFT matrix
- If $f$ has only one term, the problem is trivial (primality testing).

# Perfect Power Evaluation Witnesses

## Theorem

*Suppose* $f \in \mathbb{F}_q[x]$ *has degree* $n$ *and is not a perfect* $r^{\text{th}}$ *power. Then, if* $q \geq 4n^2$,

$$\#\left\{\alpha \in \mathbb{F}_q : f(\alpha) \text{ is an } r^{\text{th}} \text{ power}\right\} \leq \frac{3q}{4}.$$

- Means that at least $1/4$ of elements will be “good witnesses”.
- Proof uses method of completing the character sum, relying on a theorem of Weil (1948).
- No information on the *distribution* of good witnesses.

# Perfect Power Detection

**Theorem**

*For $f \in \mathbb{Z}[x]$ with degree $n$ and $t$ terms, we can determine whether $f$ is a perfect power using $O(t \log^2 \|f\|_\infty \log^2 n)$ bit operations.*

- Probabilistic Monte Carlo algorithm
  (always fast; correct with arbitrarily high probability)
- For $f \in \mathbb{Q}[x]$, can reduce to $\mathbb{Z}[x]$ by Gauß's Lemma
- The case of $f \in \mathbb{F}_q[x]$ is already handled!
- For $f \in \mathrm{R}[x_1, x_2, \ldots, x_k]$, substitute random values for $x_2, \ldots, x_k$ and work over $\mathrm{R}[x_1]$.

# Previously-Known Methods to Detect Perfect Powers

## Square-Free Decomposition (Yun '76)

Computes $f = g_1^{d_1} g_2^{d_2} \cdots g_k^{d_k}$,
for square-free, relatively prime, nonconstant $g_1, \ldots, g_k$.
$f$ is a perfect power iff $\gcd(d_1, \ldots, d_k) > 1$.

## Newton Iteration

For each $r$, computes a power series $r^{\text{th}}$ root of $f$.
Then Monte Carlo check by random evaluation.

- Follows best method for computing roots of integers
  (Bach & Sorenson '93; Bernstein '98)

# Previously-Known Methods to Detect Perfect Powers

## Square-Free Decomposition (Yun '76)

Computes $f = g_1^{d_1} g_2^{d_2} \cdots g_k^{d_k}$,
for square-free, relatively prime, nonconstant $g_1, \ldots, g_k$.
$f$ is a perfect power iff $\gcd(d_1, \ldots, d_k) > 1$.
Implemented natively in NTL

## Newton Iteration

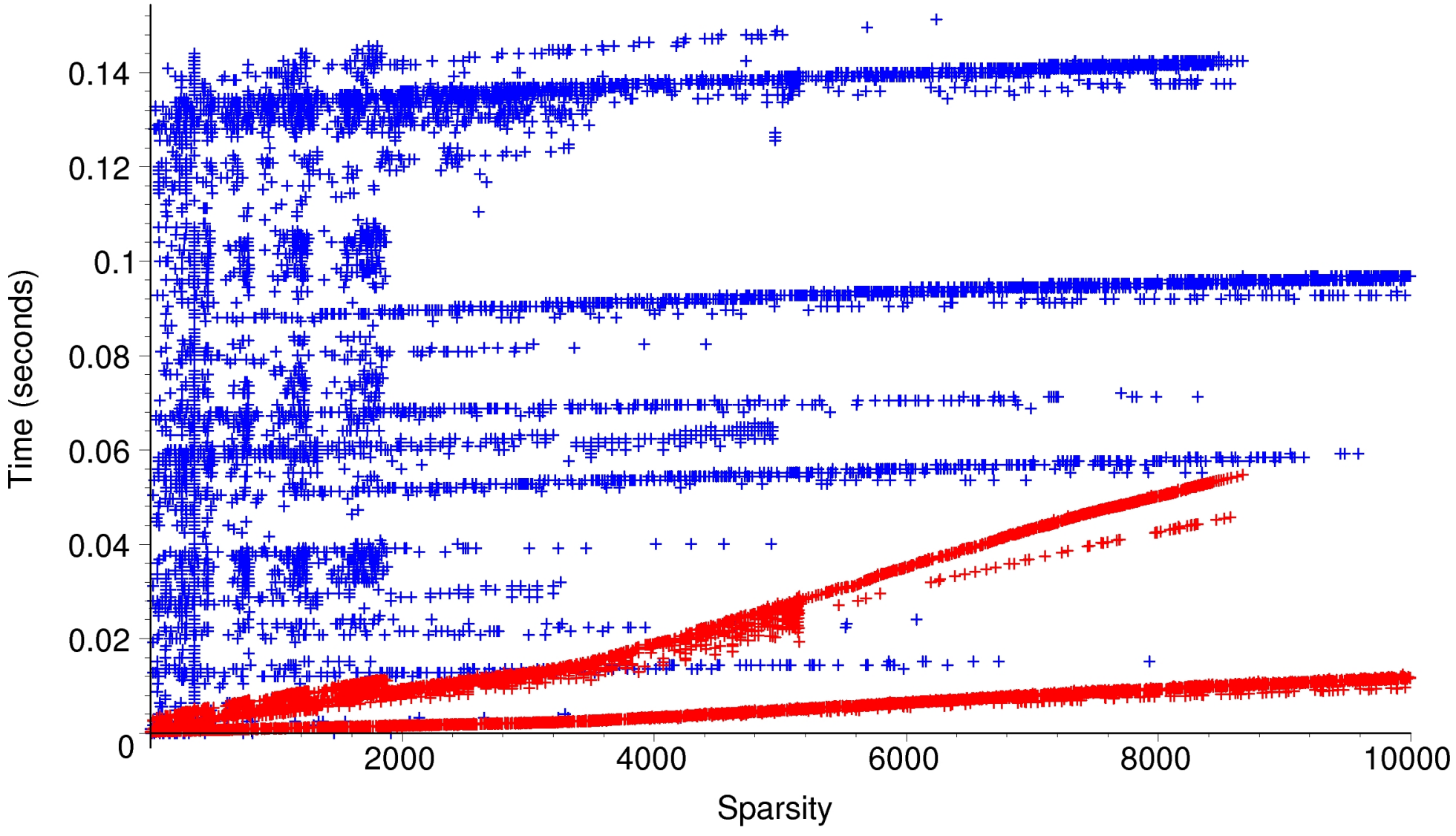
For each $r$, computes a power series $r^{\text{th}}$ root of $f$.
Then Monte Carlo check by random evaluation.
Implemented in NTL by us

- Follows best method for computing roots of integers
  (Bach & Sorenson '93; Bernstein '98)

We also implemented our detection algorithm in NTL.

# Notes on the Timings

- Square-free decomposition was always much slower; not shown
- Millions of trials, ZERO failures
- Our algorithm wins even for dense polynomials (it's really a black-box algorithm)
- Not a fair comparison because we're not computing the root (yet. . . )


Newton iteration (blue) vs. Lacunary Alg. (red) for $n = 10,000$
Time (seconds)
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
2000
4000
6000
8000
10000
Sparsity

# Computing Perfect $r^{\text{th}}$ roots

## Problem

We can determine if $f = h^r$ for some $h$, and find $r$, but how do we compute $h$ given $r$ and (lacunary) $f$?

How dense can the root be?

- This is a very old, well-studied problem
  Erdös ('47); Coppersmith & Davenport ('91); Abbot ('02); Schinzel ('87); Zannier ('07)
- Still not known whether a sparse poly. may have a dense root.
  (Our first thm. proves this for fixed-height integer polynomials)
- We seek only output-sensitive algorithms
  (i.e. assume $h$ is sparse)

# Newton Iteration to Compute Roots

**Input:** $f \in \mathrm{R}[x], r \in \mathbb{N}$ s.t. $f$ is a perfect $r^{\text{th}}$ power
**Output:** $h \in \mathrm{R}[x]$ s.t. $f = h^r$

- $k \leftarrow 1; h \leftarrow \sqrt[r]{f(0)}$
- **while** $k \leq (\deg f)/r$
  - $h \leftarrow h + \dfrac{f - h^r}{r\,h^{r-1}} \mod x^{2k}$
  - $k \leftarrow 2k$
- **end while**

Let's rearrange to give more insight into the computation

# Newton Iteration to Compute Roots

**Input:** $f \in \mathrm{R}[x], r \in \mathbb{N}$ s.t. $f$ is a perfect $r^{\text{th}}$ power
**Output:** $h \in \mathrm{R}[x]$ s.t. $f = h^r$

- $k \leftarrow 1; h \leftarrow \sqrt[r]{f(0)}$
- **while** $k \leq (\deg f)/r$
  - $a \leftarrow \dfrac{f - h^r \bmod x^{2k}}{r x^k}$
  - $h \leftarrow h + \left(\dfrac{a}{h^{r-1}} \bmod x^k\right) \cdot x^k$
  - $k \leftarrow 2k$
- **end while**

Problem: $h^r \bmod x^{2k}$ could be dense

# Sparse Newton Iteration to Compute Roots

**Input:** $f \in \mathrm{R}[x], r \in \mathbb{N}$ s.t. $f$ is a perfect $r^{\text{th}}$ power
**Output:** $h \in \mathrm{R}[x]$ s.t. $f = h^r$

- $k \leftarrow 1; h \leftarrow \sqrt[r]{f(0)}$
- **while** $k \leq (\deg f)/r$
  - $a \leftarrow \dfrac{fh - h^{r+1} \bmod x^{2k}}{rx^k}$
  - $h \leftarrow h + \left(\dfrac{a}{f} \bmod x^k\right) \cdot x^k$
  - $k \leftarrow 2k$
- **end while**

But we can prove $h^{r+1} \bmod x^{2k}$ is *not* dense (assuming the output is sparse).

# Complexity of Computing the Root

Need a conjecture to prove output-sensitive polynomial time:

## Conjecture

Given $h \in \mathrm{R}[x]$ and $r, k \in \mathbb{N}$, we can compute $h^r \bmod x^k$ in time polynomial in:

- The lacunary size of $h$
- The lacunary size of $h^r \bmod x^k$, and
- $\log r$

- Repeated squaring and truncating seems to satisfy this.

## Update

We now have a (different) provable method of computing $r^{\text{th}}$ roots in output-sensitive polynomial time
(almost certainly less efficient in practice).

# Conclusions and Future Directions

## Contributions

- Monte Carlo polynomial-time algorithm to determine if a lacunary polynomial is a perfect power
- Sparsity-sensitive Newton iteration to compute the root (subject to conjectures)

## Future Directions

**Other Models** Straight-line programs, black boxes, . . .

**Other Domains** Sparse integers, approximate, . . .

**Other Problems** Divisibility, reducibility, factorization, . . .

# Some Open Problems

- Given $f \in \mathbb{Z}[x]$ with $t$ nonzero terms, what is the *least* number of terms in $f^2$? $f^r$? $g \circ f$?
- Given a black box (or SLP) for $f \in \mathbb{F}_q[x]$, construct a black box (or SLP) for $f^2 \bmod x^k$.
- Find polynomial-time algorithms for lacunary polynomial divisibility or reducibility tests, or prove they are NP-hard.
- Solve any of these problems for sparse integers.