Detecting Polynomial Perfect Powers

Mark Giesbrecht
Daniel S. Roche

Symbolic Computation Group
University of Waterloo
Waterloo, Ontario, Canada

1 Lacunary Polynomials

We seek algorithms that are fast when the input is given in the **lacunary representation**:

For $f \in \mathsf{R}[x_1, \ldots, x_\ell]$ with total degree n given by

$$f(x) = c_1\mathbf{x}^{\mathbf{e}_1} + c_2\mathbf{x}^{\mathbf{e}_2} + \cdots + c_t\mathbf{x}^{\mathbf{e}_t},$$

where each $c_i \in \mathsf{R} \setminus \{0\}$ and $\mathbf{x}^{\mathbf{e}_i} = x_1^{(\mathbf{e}_i)_1} x_2^{(\mathbf{e}_i)_2} \cdots x_\ell^{(\mathbf{e}_i)_\ell}$, we store only a linked list of tuples $(c_i, \mathbf{e}_i)$, for a total size of $O(t\ell \log n)$.

So by "fast", we mean complexity polynomial in $t\ell \log n$, and in the case that $\mathsf{R} = \mathbb{Q}$, polynomial in the height of f as well.

Why worry about this representation?

- Corresponds to intuition and how humans express polynomials
- Default representation in Maple, Mathematica, etc.
- Can be **exponentially** smaller than dense representation

For problems with fast algorithms in the dense representation,
- some are intractable for lacunary polynomials (gcd, factoring, squarefreeness),
- some have fast but different algorithms for lacunary polynomials (interpolation, finding roots or "small" factors),
- and for some we don't know yet (irreducibility, divisibility testing)

2 Are You Perfect?

Question. *Given a multivariate polynomial $f \in \mathsf{R}[x_1, \ldots, x_\ell]$, how do we determine if f is a perfect power?*

That is, are there $h \in \mathsf{R}[x_1, \ldots, x_\ell]$ and $r \geq 2$ such that $f = h^r$?

- Detecting multi-precision integer perfect powers was studied by Bach and Sorenson (1993), later by others (most recently Bernstein et al. (2007)).
- Strong relationships to factoring and irreducibility testing
- Detecting sparse integer and lacunary polynomial perfect powers stated as open problems by Shparlinski (2000).

3 Summary of Results

For multivariate polynomials over $\mathbb{Q}$ or a finite field with sufficiently large characteristic, we present *Monte Carlo* algorithms to detect lacunary polynomials which are perfect powers.

That is, our algorithms are correct with controllably high probability and always fast.

4 Integer Polynomials Algorithm

Input: $f \in \mathbb{Z}[x]$
Output: An r such that f is an rth power,
or "FALSE" if f is not a perfect power.

1. **for** each possible prime power r **do**
2. Pick a prime p with $p \nmid \operatorname{disc}(f)$
3. Pick a prime power q with $r \mid q - 1$
4. Choose random $\alpha_1, \ldots, \alpha_5 \in \mathbb{F}_q$
5. **if** $f(\alpha_1)^{(q-1)/r} = \cdots = f(\alpha_5)^{(q-1)/r} = 1$ over $\mathbb{F}_q$
6. **return** r
7. **end do**
8. **return** "FALSE"

Boring Details

- Steps 2–6 will actually be repeated $O(\log 1/\epsilon)$ times to guarantee success with probability $1 - \epsilon$.
- For Step 3, we can either find a random prime q such that $p \mid q$ and $r \mid (q-1)$, or choose $q = p^{r-1}$ and work in an extension field. The first approach yields better practical performance but poorer theoretical results.
- If $f \in \mathbb{F}_\varrho$ for some prime power ϱ, the same algorithm will work, replacing p with ϱ and omitting Step 2.
- For $f \in \mathbb{Q}[x]$, choose the smallest $b \in \mathbb{N}$ such that $bf \in \mathbb{Z}[x]$. Then bf is a perfect power iff f is, so we run the algorithm on input bf.
- For $f \in \mathsf{R}[x_1, \ldots, x_\ell]$, choose random values $\beta_2, \ldots, \beta_\ell \in \mathsf{R}$, and then test whether $f(x, \beta_2, \ldots, \beta_\ell) \in \mathsf{R}[x]$ is a perfect power.

5 Why it works

Reduction: Since $p \nmid \operatorname{disc}(f)$, f is an rth power over $\mathbb{Z}[x]$ iff f is an rth power over $\mathbb{F}_p[x]$.

Detection: $f(\alpha_i)^{(q-1)/r} = 1$ iff $f(\alpha_i)$ is an rth power in $\mathbb{F}_q$.

Implication: Clearly if $f = h^r$ for some h and $r \geq 2$, then each $f(\alpha)$ is a perfect rth power in $\mathbb{F}_q$.

The other direction is more interesting:

Theorem. *Suppose $f \in \mathbb{F}_q[x]$ is not a perfect rth power, and the degree of f is not more than $1 + \sqrt{q}/2$. Then, for a random $\alpha \in \mathbb{F}_q$, the probability that $f(\alpha)$ is a perfect rth power in $\mathbb{F}_q$ is less than $3/4$.*

The proof uses an exponential character sum argument and the powerful Weil's Theorem for character sums with polynomial arguments. Since $(3/4)^5 < 1/4$, choosing 5 random evaluations guaranees success with at least 3/4 probability.

6 How high can the power be?

- Speed of algorithm depends on how many r's and how big
- Number of r's is $O(\log \deg f)$ since all are distinct prime divisors of $\deg f$
- But can r be large?

Schinzel (1987) gives the following upper bound on r:

Fact. *For $f \in \mathsf{F}[x]$ with t nonzero terms and $\deg f$ less than the characteristic of F, $r \leq t - 1$.*

We have the following stronger result for integer polynomials:

Theorem. *If $f, h \in \mathbb{Z}[x]$ such that $f = h^r$, then $\|h\|_2 \leq \|f\|_1^{1/r}$.*

For the proof, first note that the average value of $|h(\theta)|^2$ for a primitive pth root of unity θ (where $p > \deg h$) is $\|h\|_2^2$. Then there exists a θ with $|\theta| = 1$ s.t. $|h(\theta)| \geq \|h\|_2$. Therefore

$$\|h\|_2 \leq |h(\theta)| = |f(\theta)|^{1/r} \leq \|f\|_1^{1/r}.$$

Since $\|h\|_2 \geq \sqrt{2}$ in all nontrivial cases, this means $r \leq 2\log_2 \|f\|_1$.

7 Complexity

How big is p? $\operatorname{disc}(f) = \operatorname{res}(f, f') \in O(n(\log n + \log \|f\|_2))$, so a prime with $O(\log n + \log\log \|f\|_\infty)$ bits does not divide the discriminant with high probability.

How many operations in $\mathbb{F}_q$? The most costly step is computing each $f(\alpha_i)^{(q-1)/r}$ at Step 5. Computing each $f(\alpha_i)$ can be accomplished using $O(t \log n)$ operations in $\mathbb{F}_q$, and then these evaluations can be raised to the power $(q-1)/r$ with an additional $O(\log q - \log r)$ operations in $\mathbb{F}_q$.

How costly are operations in $\mathbb{F}_q$? If we choose $q = p^{r-1}$ and work in a field extension modulo an irreducible polynomial in $\mathbb{F}_p$ of degree $r - 1$, then each operation in $\mathbb{F}_q$ will cost $O\tilde{}(r \log p)$ bit operations, which is $O\tilde{}(r(\log n + \log\log \|f\|_\infty))$.

Total Bit Complexity Finally, beacuse $r \in O(\log(t\|f\|_\infty))$, we have a total bit complexity of $O\tilde{}(t \log^2 \|f\|_\infty \log^2 n)$, which is polynomial in the lacunary size of f, as desired.

8 Implementation

We used Victor Shoup's NTL to implement and compare the performance of the following three algorithms. This is a C++ library which, when coupled with GMP, provides (probably) the fastest implementations of arithmetic for dense univariate polynomials, multi-precision integers, and finite fields.

Algorithms to Compare

Square-Free Decomposition For $f \in \mathbb{Z}[x]$, computes $d_1, \ldots, d_s \in \mathbb{N}$ and squarefree $g_1, \ldots, g_s \in \mathbb{Z}[x] \setminus \mathbb{Z}$ such that $f = g_1^{d_1} g_2^{d_2} \cdots g_s^{d_s}$. So f is a perfect power iff $\gcd(d_1, \ldots, d_s) > 1$.

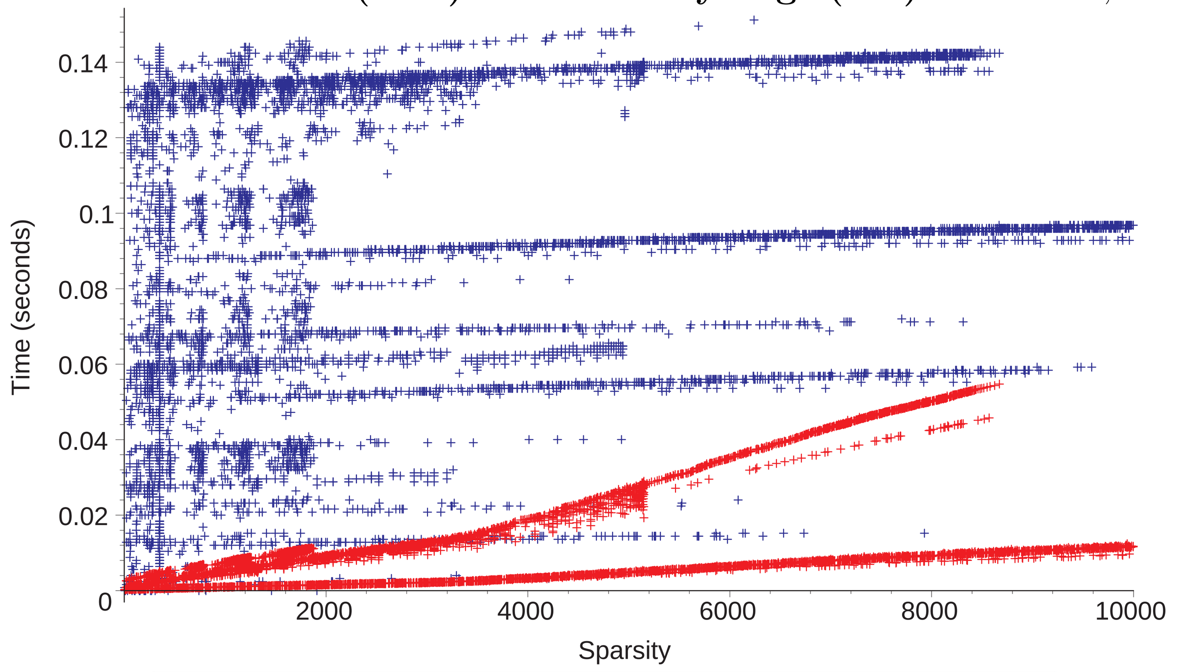
Newton Iteration Given $f \in \mathsf{F}[x]$ and $r \in \mathbb{N}$, there is a unique $h \in \mathsf{F}[x]$ such that $h^r \equiv f \bmod x^{n/r}$. For each possible r, we compute such an h using a Newton Iteration taking $\log_2(n/r)$ steps, and then check whether $h(\alpha)^r = f(\alpha)$ for a randomly-chosen $\alpha \in \mathsf{F}$. The result is a Monte Carlo algorithm similar to the one above.

Our Algorithm We also implemented the lacunary algorithm described above. As NTL does not provide sparse polynomial arithmetic, we had to implement a procedure to quickly evaluate a lacunary polynomial at a given point, using NTL for the underlying field arithmetic.

Some timings for Square-Free Decomposition (black), Newton Iteration (blue), and Lacunary Algorithm (red)

	$n = 1,000$				$n = 10,000$				$n = 100,000$			
	$t \approx 500$		$t = n+1$		$t \approx 500$		$t = n+1$		$t \approx 500$		$t = n+1$	
	$r=1$	$r=5$	$r=1$	$r=5$	$r=1$	$r=5$	$r=1$	$r=5$	$r=1$	$r=5$	$r=1$	$r=5$
S	.0086	.0159	.0065	.0194	.2872	.2422	.1329	.3368	6.458	5.459	2.925	7.301
N	.0029	.0035	.0034	.0046	.0557	.0643	.0848	.1060	1.183	1.432	1.544	1.971
L	.0003	.0022	.0003	.0029	.0004	.0030	.0016	.0154	.0005	.0040	.0174	.1582

Newton iteration (blue) vs. Lacunary Alg. (red) for $n = 10,000$



Timing Comparisons

References

E. Bach and J. Sorenson. Sieve algorithms for perfect power testing. *Algorithmica*, 9(4):313–328, 1993.

D. Bernstein, H.W. Lenstra, Jr., and J. Pila. Detecting perfect powers by factoring into coprimes. *Math. Comp.*, 76(257):385–388 (electronic), 2007. ISSN 0025-5718.

M. Giesbrecht and D. Roche. On lacunary polynomial perfect powers. In *ISSAC '08 (submitted)*, 2008.

A. Schinzel. On the number of terms of a power of a polynomial. *Acta Arith.*, 49(1):55–70, 1987. ISSN 0065-1036.

I. Shparlinski. Computing Jacobi symbols modulo sparse integers and polynomials and some applications. *J. Algorithms*, 36(2):241–252, 2000. ISSN 0196-6774.