# Detecting lacunary perfect powers and computing their roots

Mark Giesbrecht

*Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada*

Daniel S. Roche

*Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada*

**Abstract**

We consider the problem of determining whether a lacunary (also called a *sparse* or *super-sparse*) polynomial $f$ is a perfect power, that is, $f = h^r$ for some other polynomial $h$ and $r \in \mathbb{N}$, and of finding $h$ and $r$ should they exist. We show how to determine if $f$ is a perfect power in time polynomial in the number of non-zero terms of $f$, and in terms of $\log \deg f$, i.e., polynomial in the size of the lacunary representation. The algorithm works over $\mathbb{F}_q[x]$ (for large characteristic) and over $\mathbb{Z}[x]$, where the cost is also polynomial in $\log \|f\|_\infty$. We also give a Monte Carlo algorithm to find $h$ if it exists, for which our proposed algorithm requires polynomial time in the output size, i.e., the sparsity and height of $h$. Conjectures of Erdös and Schinzel, and recent work of Zannier, suggest that $h$ must be sparse. Subject to a slightly stronger conjectures we give an extremely efficient algorithm to find $h$ via a form of sparse Newton iteration. We demonstrate the efficiency of these algorithms with an implementation using the `C++` library NTL.

## 1. Introduction

In this paper we consider the problem of determining whether a polynomial $f$ equals $h^r$ for some other polynomial $h$ and positive integer $r$, and if so, finding $h$ and $r$. The novel aspect of this current work is that our algorithms are efficient for the *lacunary* (also called *sparse* or *supersparse*) representation of polynomials. Specifically, we assume

$$f = \sum_{1 \le i \le t} c_i \overline{x}^{\overline{e}_i} \in \mathsf{F}[x_1, \ldots, x_\ell], \tag{1.1}$$

where $\mathsf{F}$ is a field, $c_0, \ldots, c_t \in \mathsf{F} \backslash \{0\}$, and $\overline{e}_1, \ldots, \overline{e}_t \in \mathbb{N}^\ell$ are distinct exponent tuples with $0 \le \|\overline{e}_1\|_1 \le \cdots \le \|\overline{e}_t\|_1 = \deg f$, where $\overline{x}_i^{\overline{e}_i}$ is the monomial $x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_\ell^{e_{i\ell}}$ of degree

*Email addresses:* `mwg@uwaterloo.ca` (Mark Giesbrecht), `droche@cs.uwaterloo.ca` (Daniel S. Roche).

$\|\overline{e}_i\|_1 = \sum_{1\le j\le \ell} e_{ij}$. We say $f$ is *t-sparse* and write $\tau(f) = t$. We present algorithms which require time polynomial in $\tau(f)$ and $\log \deg f$.

Computational work on *lacunary* polynomials has proceeded steadily for the past three decades. From the dramatic initial intractability results of Plaisted (1977, 1984), through progress in algorithms (e.g., Ben-Or and Tiwari (1988), Shparlinski (2000), and Kaltofen and Lee (2003)) and complexity (e.g., Karpinski and Shparlinski (1999), Quick (1986), and von zur Gathen et al. (1993)), to recent breakthroughs in root finding and factorization (Cucker et al., 1999; Kaltofen and Koiran, 2006; Lenstra, 1999), these works have important theoretical and practical consequences. The lacunary representation is arguably more intuitive than the standard dense representation, and in fact corresponds to the default linked-list representation of polynomials in modern computer algebra systems such as Maple and Mathematica.

We will always assume that $\tau(f) \ge 2$; otherwise $f = x^n$, and determining whether $f$ is a perfect power is equivalent to determining whether $n \in \mathbb{N}$ is composite, and to factoring $n$ if we wish to produce $r$ dividing $n$ such that $f = (x^{n/r})^r$. Surprisingly, the intractability of the latter problem is avoided when $\tau(f) \ge 2$.

We first consider detecting perfect powers and computing the power $r$ for the univariate case

$$f = \sum_{1\le i\le t} c_i x^{e_i} \in \mathsf{F}[x], \tag{1.2}$$

where $0 \le e_1 < e_2 < \cdots < e_t = \deg f$.

Two cases for the field $\mathsf{F}$ are handled: the integers and finite fields of characteristic $p$ greater than the degree of $f$. When $f \in \mathbb{Z}[x]$, our algorithms also require time polynomial in $\log \|f\|_\infty$, where $\|f\|_\infty = \max_{1\le i\le t} |c_i|$ (for $f \in \mathbb{Q}[x]$, we simply work with $\overline{f} = cf \in \mathbb{Z}[x]$, for the smallest $c \in \mathbb{Z}\backslash\{0\}$). This reflects the bit-length of coefficients encountered in the computations. Efficient techniques will also be presented for reducing the multivariate case to the univariate one, and for computing a root $h$ such that $f = h^r$.

### *1.1. Related work and methods*

Two well-known techniques can be applied to the problem of testing for perfect powers, and both are very efficient when $f = h^r$ is dense. We can compute the squarefree decomposition of $f$ as in (Yun, 1976), and determine whether $f$ is a perfect power by checking whether the greatest (integer) common divisor of the exponents of all non-trivial factors in the squarefree decomposition is at least 2. An even faster method (in theory and practice) to find $h$ given $f = h^r$ is by a Newton iteration. This technique has also proven to be efficient in computing perfect roots of (dense) multi-precision integers (Bach and Sorenson, 1993; Bernstein, 1998). In summary however, we note that both these methods require approximately linear time in the *degree* of $f$, which may be exponential in the lacunary size.

Newton iteration has also been applied to finding perfect polynomial roots of lacunary (or other) polynomials given by straight-line programs. Kaltofen (1987) shows how to compute a straight-line program for $h$, given a straight-line program for $f = h^r$ and the value of $r$. This method has complexity polynomial in the size of the straight-line program for $f$, and in the degree of $h$, and in particular is effective for large $r$. We do not address the powerful generality of straight-line programs, but do avoid the dependence on the degree of $h$.

Closest to this current work, Shparlinski (2000) shows how to recognize whether $f = h^2$ for a lacunary polynomial $f \in \mathbb{F}_q[x]$. Shparlinski uses random evaluations and tests for quadratic residues. How to determine whether a lacunary polynomial is *any* perfect power is posed as an open question.

### *1.2. Our contributions*

Given a lacunary polynomial $f \in \mathbb{Z}[x]$ with $\tau(f) \geq 2$ and degree $n$, we first present an algorithm to compute an integer $r > 1$ such that $f = h^r$ for some $h \in \mathbb{Z}[x]$, or determine that no such $r$ exists. The algorithm requires $O^{\sim}(t \log^2 \|f\|_\infty \log^2 n)$ machine operations*, and is probabilistic of the Monte Carlo type. That is, for any input, on any execution the probability of producing an incorrect answer is strictly less than $1/2$, assuming the ability to generate random bits at unit cost. This possibility of error can be made arbitrarily small with repeated executions.

A similar algorithm is presented to answer Shparlinski's open question on perfect powers of lacunary polynomials over finite fields, at least for the case of large characteristic. That is, when the characteristic $p$ of a finite field $\mathsf{F}$ is greater than $\deg f$, we provide a Monte Carlo algorithm that determines if there exists an $h \in \mathsf{F}[x]$ and $r$ such that $f = h^r$, and finds $r$ if it exists, which requires $O^{\sim}(t \log^2 n)$ operations in $\mathsf{F}$.

An implementation of our algorithm over $\mathbb{Z}$ in NTL indicates excellent performance on sparse inputs when compared to a fast implementation based on previous technology (a variable-precision Newton iteration to find a power-series $r$th root of $f$, followed by a Monte Carlo correctness check).

Actually computing $h$ such that $f = h^r$ is a somewhat trickier problem, at least insofar as bounds on the sparsity of $h$ have not been completely resolved. Conjectures of Schinzel (1987) and recent work of Zannier (2007) suggest that, provided the characteristic of $\mathsf{F}$ is zero or sufficiently large, $h$ is lacunary as well. To avoid this lack of sufficient theoretical understanding, we develop an algorithm which requires time polynomial in both the representation size of the input $f$ (i.e., $\tau(f)$, $\log n$ and $\log \|f\|_\infty$) *and* the representation size of the output (i.e., $\tau(h)$ and $\log \|f\|_\infty$). This algorithm works by projecting $f$ into a sequence of small cyclotomic fields. Images of the desired $h$ in these fields are discovered by factorization over an algebraic extension. Finally, a form of interpolation of the sparse exponents is used to recover the global $h$. The algorithm is probabilistic of the Monte Carlo type. While this algorithm is polynomial time, we do not claim it will be efficient in practice. Instead, we also present and analyze a simpler alternative based on a kind of Newton iteration. Subject to what we believe is a reasonable conjecture, this is shown to be very fast.

The remainder of the paper is arranged as follows. In Section 2 we present the main theoretical tool for our algorithm to determine if $f = h^r$, and to find $r$. We also show how to reduce the multivariate problem to the univariate one. In Section 3 we show how to compute $h$ such that $f = h^r$ (given that such $h$ and $r$ exist). Finally, in Section 4, we present an experimental implementation of some of our algorithms in the `C++` library NTL.

An earlier version of some of this work was presented in the ISSAC 2008 conference (Giesbrecht and Roche, 2008).

---

* We employ soft-Oh notation: for functions $\sigma$ and $\varphi$ we say $\sigma \in O^{\sim}(\varphi)$ if $\sigma \in O(\varphi \log^c \varphi)$ for some constant $c > 0$.

## 2. Testing for perfect powers

In this section we describe a method to determine if a lacunary polynomial $f \in \mathsf{F}[x]$ is a perfect power. That is, do there exist $h \in \mathsf{F}[x]$ and $r > 1$ such that $f = h^r$? The polynomial $h$ need not be lacunary, though some conjectures suggest it may well have to be. We will find $r$, but not $h$.

We first describe algorithms to test if an $f \in \mathsf{F}[x]$ is an $r$th power of some polynomial $h \in \mathsf{F}[x]$, where $f$ and $r$ *are both given* and $r$ is assumed to be prime. We present and analyze variants that work over finite fields $\mathbb{F}_q$ and over $\mathbb{Z}$. In fact, these algorithms for given $r$ are for *black-box* polynomials: they only need to evaluate $f$ at a small number of points. That this evaluation can be done quickly is a property of lacunary and other classes of polynomials.

For lacunary $f$ we then show that, in fact, if $h$ exists at all then $r$ must be small unless $f = x^n$. And if $f$ is a perfect power, then there certainly exists a prime $r$ such that $f$ is an $r$th power. So in fact the restrictions that $r$ is small and prime are sufficient to cover all nontrivial cases, and our method is complete.

### *2.1. Detecting given rth powers*

Our main tool in this work is the following theorem which says that, with reasonable probability, a polynomial is an $r$th power if and only if the modular image of an evaluation in a specially constructed finite field is an $r$th power.

**Theorem 2.1.** Let $\varrho \in \mathbb{Z}$ be a prime power and $r \in \mathbb{N}$ a prime dividing $\varrho - 1$. Suppose that $f \in \mathbb{F}_\varrho[x]$ has degree $n \le 1 + \sqrt{\varrho}/2$ and is *not* a perfect $r$th power in $\mathbb{F}_\varrho[x]$. Then

$$R_f^{(r)} = \#\{c \in \mathbb{F}_\varrho : f(c) \in \mathbb{F}_\varrho \text{ is an } r\text{th power}\} \le \frac{3\varrho}{4}.$$

**Proof.** The $r$th powers in $\mathbb{F}_\varrho$ form a subgroup $H$ of $\mathbb{F}_\varrho^*$ of index $r$ and size $(\varrho-1)/r$ in $\mathbb{F}_\varrho^*$. Also, $a \in \mathbb{F}_\varrho^*$ is an $r$th power if and only if $a^{(\varrho-1)/r} = 1$. We use the method of "completing the sum" from the theory of character sums. We refer to Lidl and Niederreiter (1983), Chapter 5, for an excellent discussion of character sums. By a multiplicative character we mean a homomorphism $\chi : \mathbb{F}_\varrho^* \to \mathbb{C}$ which necessarily maps $\mathbb{F}_\varrho$ onto the unit circle. As usual we extend our multiplicative characters $\chi$ so that $\chi(0) = 0$, and define the trivial character $\chi_0(a)$ to be 0 when $a = 0$ and 1 otherwise.

For any $a \in \mathbb{F}_\varrho^*$,

$$\frac{1}{r} \sum_{\chi^r = \chi_0} \chi(a) = \begin{cases} 1 & \text{if } a \in H, \\ 0 & \text{if } a \notin H, \end{cases}$$

where $\chi$ ranges over all the multiplicative characters of order $r$ on $\mathbb{F}_\varrho^*$ — that is, all characters that are isomorphic to the trivial character on the subgroup $H$. Thus

$$\begin{aligned} R_f^{(r)} &= \sum_{a \in \mathbb{F}_\varrho^*} \left( \frac{1}{r} \sum_{\chi^r = \chi_0} \chi(f(a)) \right) = \frac{1}{r} \sum_{\chi^r = \chi_0} \sum_{a \in \mathbb{F}_\varrho^*} \chi(f(a)) \\ &\le \frac{\varrho}{r} + \frac{1}{r} \sum_{\substack{\chi^r = \chi_0 \\ \chi \ne \chi_0}} \left| \sum_{a \in \mathbb{F}_\varrho} \chi(f(a)) \right|. \end{aligned}$$

Here we use the obvious fact that

$$\sum_{a\in\mathbb{F}_\varrho^*}\chi_0(f(a)) \le \sum_{a\in\mathbb{F}_\varrho}\chi_0(f(a)) = \varrho - d \le \varrho,$$

where $d$ is the number of distinct roots of $f$ in $\mathbb{F}_\varrho$. We next employ the powerful theorem of Weil (1948) on character sums with polynomial arguments (see Theorem 5.41 of Lidl and Niederreiter (1983)), which shows that if $f$ is *not* a perfect $r$th power of another polynomial, and $\chi$ has order $r > 1$, then

$$\left|\sum_{a\in\mathbb{F}_\varrho}\chi(f(a))\right| \le (n-1)\varrho^{1/2} \le \frac{\varrho}{2},$$

using the fact that we insisted $n \le 1+\sqrt{\varrho}/2$. Summing over the $r-1$ non-trivial characters of order $r$, we deduce that

$$R_f^{(r)} \le \frac{\varrho}{r} + \frac{r-1}{r}\cdot\frac{\varrho}{2} \le \frac{3\varrho}{4}. \qquad \square$$

### *2.2. Certifying specified powers over $\mathbb{F}_q[x]$*

Theorem 2.1 allows us to detect when a polynomial $f \in \mathbb{F}_\varrho[x]$ is a perfect $r$th power, for known $r$ dividing $\varrho - 1$: choose random $\alpha \in \mathbb{F}_\varrho$ and evaluate $\xi = f(\alpha)^{(\varrho-1)/r} \in \mathbb{F}_\varrho$. Recall that $\xi = 1$ if and only if $f(\alpha)$ is an $r$th power.

- If $f$ is an $r$th power, then clearly $f(\alpha)$ is an $r$th power and we always have $\xi = 1$.
- If $f$ is not an $r$th power, Theorem 2.1 demonstrates that for at least $1/4$ of the elements of $\mathbb{F}_\varrho$, $f(\alpha)$ is not an $r$th power. Thus, for $\alpha$ chosen randomly from $\mathbb{F}_\varrho$ we would expect $\xi \ne 1$ with probability at least $1/4$.

For a polynomial $f \in \mathbb{F}_q[x]$ over an arbitrary finite field $\mathbb{F}_q$, where $q$ is a prime power such that $q-1$ is not divisible by $r$, we proceed by constructing an extension field $\mathbb{F}_{q^{r-1}}$ over $\mathbb{F}_q$. From Fermat's Little Theorem and the fact that $r \nmid q$, we know $r \mid (q^{r-1}-1)$, and we can proceed as above. We now present and analyze this more formally.

**Algorithm** `IsPerfectRthPowerGF`

**Input:** A prime power $q$, $f \in \mathbb{F}_q[x]$ of degree $n \le 1+\sqrt{q}/2$, $r \in \mathbb{N}$ a prime dividing $n$, and $\epsilon \in \mathbb{R}_{>0}$

**Output:** True if $f$ is the $r$th power of a polynomial in $\mathbb{F}_\varrho[x]$; False otherwise.

1: Find an irreducible $\Gamma \in \mathbb{F}_q[z]$ of degree $r-1$, successful with probability at least $\epsilon/2$
2: $\varrho \leftarrow q^{r-1}$
3: Define $\mathbb{F}_\varrho = \mathbb{F}_q[z]/(\Gamma)$
4: $m \leftarrow 2.5(1+\lceil\log_2(1/\epsilon)\rceil)$
5: **for** $i$ from 1 to $m$ **do**
6: Choose random $\alpha \in \mathbb{F}_\varrho$
7: $\xi \leftarrow f(\alpha)^{(\varrho-1)/r} \in \mathbb{F}_\varrho$
8: **if** $\xi \ne 1$ **then**
9: **return** False
10: **return** True

**Notes on** `IsPerfectRthPowerGF`**.**

To accomplish Step 1, a number of fast probabilistic methods are available to find irreducible polynomials. We employ the algorithm of Shoup (1994). This algorithm requires $O((r^2 \log r + r \log q) \log r \log\log r)$ operations in $\mathbb{F}_q$. It is probabilistic of the Las Vegas type, and we assume that it always stops within the number of operations specified, and returns the correct answer with probability at least $1/2$ and "Fail" otherwise (it never returns an incorrect answer). The algorithm is actually presented in Shoup (1994) as *always* finding an irreducible polynomial, but requiring *expected* time as above; by not iterating indefinitely our restatement allows for a Monte Carlo analysis in what follows. To obtain an irreducible $\Gamma$ with failure probability at most $\epsilon/2$ we run (our modified) Shoup's algorithm $1 + \lceil \log_2(1/\epsilon) \rceil$ times.

The restriction that $n \leq 1 + \sqrt{2}$ (or alternatively $q \geq 4(n-1)^2$) is not problematic. If this condition is not met, simply extend $\mathbb{F}_q$ with an extension of degree $\nu = \lceil \log_q(4(n-1)^2) \rceil$ and perform the algorithm over $\mathbb{F}_{q^\nu}$. At worst, each operation in $\mathbb{F}_{q^\nu}$ requires $O(\mathrm{M}(\log n))$ operations in $\mathbb{F}_q$.

Here we define $\mathrm{M}(r)$ as a number of operations in $\mathsf{F}$ to multiply two polynomials of degree $\leq r$ over $\mathsf{F}$, for any field $\mathsf{F}$, or the number of bit operations to multiply two integers with at most $r$ bits. Using classical arithmetic $\mathrm{M}(r)$ is $O(r^2)$, while using the fast algorithm of Cantor and Kaltofen (1991) we may assume $\mathrm{M}(r)$ is $O(r \log r \log\log r)$.

**Theorem 2.2.** Let $q$ be a prime power, $f \in \mathbb{F}_q[x]$, $r \in \mathbb{N}$ a prime dividing $\deg f$ and $\epsilon > 0$. If $f$ is a perfect $r$th power the algorithm `IsPerfectRthPowerGF` always reports this. If $f$ is not a perfect $r$th power then, on any invocation, this is reported correctly with probability at least $1 - \epsilon$.

**Proof.** It is clear from the above discussion that the algorithm always works when $f$ is perfect power. When $f$ is not a perfect power, each iteration of the loop will obtain $\xi \neq 1$ (and hence a correct output) with probability at least $1/4$. By iterating the loop $m$ times we ensure that the probability of failure is at most $\epsilon/2$. Adding this to the probability that Shoup's algorithm (for Step 1) fails yields a total probability of failure of at most $\epsilon$. □

**Theorem 2.3.** On inputs as specified, the algorithm `IsPerfectRthPowerGF` requires $O((r\mathrm{M}(r) \log r \log q) \cdot \log(1/\epsilon))$ operations in $\mathbb{F}_q$ plus the cost to evaluate $\alpha \mapsto f(\alpha)$ at $O(\log(1/\epsilon))$ points $\alpha \in \mathbb{F}_{q^{r-1}}$.

**Proof.** As noted above, Shoup's 1994 algorithm requires $O((r^2 \log r + r \log q) \log r \log\log r)$ field operations per iteration, which is within the time specified. The main cost of the loop in Steps 4–8 is computing $f(\alpha)^{(\varrho-1)/r}$, which requires $O(\log \varrho)$ or $O(r \log q)$ operations in $\mathbb{F}_\varrho$ using repeated squaring, plus one evaluation of $f$ at a point in $\mathbb{F}_\varrho$. Each operation in $\mathbb{F}_\varrho$ requires $O(\mathrm{M}(r))$ operations in $\mathbb{F}_q$, and we repeat the loop $O(\log(1/\epsilon))$ times. □

**Corollary 2.4.** Given $f \in \mathbb{F}_q[x]$ of degree $n$ with $\tau(f) = t$, and $r \in \mathbb{N}$ a prime dividing $n$, we can determine if $f$ is an $r$th power with

$$O\left((r\mathrm{M}(r) \log r \log q + t\mathrm{M}(r) \log n) \cdot \log(1/\epsilon)\right)$$

operations in $\mathbb{F}_q$. When $f$ is an $r$th power, the output is always correct, while if $f$ is not an $r$th power, the output is correct with probability at least $1 - \epsilon$.

*2.3. Certifying specified powers over $\mathbb{Z}[x]$*

For an integer polynomial $f \in \mathbb{Z}[x]$, we proceed by working in the homomorphic image of $\mathbb{Z}$ in $\mathbb{F}_p$ (and then in an extension of that field). We must ensure that the homomorphism preserves the perfect power property we are interested in with high probability. For any polynomial $g \in \mathsf{F}[x]$, let $\operatorname{disc}(g) = \operatorname{res}(f, f')$ be the discriminant of $g$ (the resultant of $f$ and its first derivative). It is well known that $g$ is squarefree if and only if $\operatorname{disc}(g) \neq 0$. Also define $\operatorname{lcoeff}(f)$ as the leading coefficient of $f$, the coefficient of the highest power of $x$ in $f$.

**Lemma 2.5.** Let $f \in \mathbb{Z}[x]$ and $\tilde{f} = f/\gcd(f, f')$ its squarefree part. Let $p$ be a prime such that $p \nmid \operatorname{disc}(\tilde{f})$ and $p \nmid \operatorname{lcoeff}(f)$. Then $f$ is a perfect power in $\mathbb{Z}[x]$ if and only if $f \bmod p$ is a perfect power in $\mathbb{F}_p[x]$.

**Proof.** Clearly if $f$ is a perfect power, then $f \bmod p$ is a perfect power in $\mathbb{Z}[x]$. To show the converse, assume that $f = f_1^{s_1} \cdots f_m^{s_m}$ for distinct irreducible $f_1, \ldots, f_m \in \mathbb{Z}[x]$, so $\tilde{f} = f_1 \cdots f_m$. Clearly $f \equiv f_1^{s_1} \cdots f_m^{s_m} \bmod p$ as well, and because $p \nmid \operatorname{lcoeff}(f)$ we know $\deg(f_i \bmod p) = \deg f_i$ for $1 \leq i \leq m$. Since $p \nmid \operatorname{disc}(\tilde{f})$, $\tilde{f} \bmod p$ is squarefree (see von zur Gathen and Gerhard (2003), Lemma 14.1), and each of the $\tilde{f_i} \bmod p$ must be pairwise relatively prime and squarefree for $1 \leq i \leq m$. Now suppose $f \bmod p$ is a perfect $r$th power modulo $p$. Then we must have $r \mid s_i$ for $1 \leq i \leq m$. But this immediately implies that $f$ is a perfect power in $\mathbb{Z}[x]$ as well. □

Given any polynomial $g = g_0 + g_1 x + \cdots + g_m x^m \in \mathbb{Z}[x]$, we define the height or coefficient $\infty$-norm of $g$ as $\|g\|_\infty = \max_i |g_i|$. Similarly, we define the coefficient 1-norm of $g$ as $\|g\|_1 = \sum_i |g_i|$, and 2-norm as $\|g\|_2 = \left(\sum_i |g_i|^2\right)^{1/2}$. Since $\tilde{f}$ divides $f$, we can employ the factor bound of Mignotte (1974) to obtain

$$\|\tilde{f}\|_\infty \leq 2^n \|f\|_2 \leq 2^n \sqrt{n} \cdot \|f\|_\infty.$$

Since $\operatorname{disc}(\tilde{f}) = \operatorname{res}(\tilde{f}, \tilde{f}')$ is the determinant of matrix of size at most $(2n-1) \times (2n-1)$, Hadamard's inequality implies

$$|\operatorname{disc}(\tilde{f})| \leq \left(2^n n^{1/2} \|f\|_\infty\right)^{n-1} \left(2^n n^{3/2} \|f\|_\infty\right)^n < 2^{2n^2} n^{2n} \cdot \|f\|_\infty^{2n}.$$

Also observe that $|\operatorname{lcoeff}(f)| \leq \|f\|_\infty$. Thus, the product $\operatorname{disc}(\tilde{f}) \cdot \operatorname{lcoeff}(f)$ has at most

$$\mu = \left\lceil \left\lceil \log_2 \left(2^{2n^2} n^{2n} \|f\|_\infty^{2n+1}\right) \right\rceil / \lfloor \log_2(4(n-1)^2) \rfloor \right\rceil$$

prime factors greater than $4(n-1)^2$ (we require the lower bound $4(n-1)^2$ to employ Theorem 2.1 without resorting to field extensions). Choose a $\gamma \geq 4(n-1)^2$ such that the number of primes $\pi(2\gamma) - \pi(\gamma)$ between $\gamma$ and $2\gamma$ is at least $4\mu + 1$. By Rosser and Schoenfeld (1962), $\pi(2\gamma) - \pi(\gamma) \geq 2\gamma/(5 \ln \gamma)$ for $\gamma \geq 59$. Thus if $\gamma \geq \max\{14\mu \ln(14\mu), 100\}$, then a random prime not equal to $r$ in the range $\gamma \ldots 2\gamma$ divides $\operatorname{lcoeff}(f) \cdot \operatorname{disc}(f)$ with probability at most $1/4$. Primes $p$ of this size have only $\log_2 p \in O(\log n + \log\log \|f\|_\infty)$ bits.

**Algorithm** `IsPerfectRthPowerZ`

**Input:** $f \in \mathbb{Z}[x]$ of degree $n$; $r \in \mathbb{N}$ a prime dividing $n$; $\epsilon \in \mathbb{R}_{>0}$;
**Output:** True if $f$ is the $r$th power of a polynomial in $\mathbb{Z}[x]$; False otherwise

1: $\mu \leftarrow \left\lceil \left\lceil \log_2 \left( 2^{2n^2} n^{2n} \|f\|_\infty^{2n+1} \right) \right\rceil / \lfloor \log_2(4(n-1)^2) \rfloor \right\rceil$
2: $\gamma \leftarrow \max\{14\mu \ln(14\mu), 4(n-1)^2, 100\}$
3: **for** $i$ from 1 to $\ldots \lceil \log_2(1/\epsilon) \rceil$ **do**
4: $\quad p \leftarrow$ random prime in the range $\gamma \ldots 2\gamma$
5: $\quad$ **if** NOT `IsPerfectRthPowerGF`($p$, $f \bmod p$, $r$, 1/4 ) **then**
6: $\quad\quad$ **return** False
7: **return** True

**Theorem 2.6.** Let $f \in \mathbb{Z}[x]$ of degree $n$, $r \in \mathbb{N}$ dividing $n$ and $\epsilon \in \mathbb{R}_{>0}$. If $f$ is a perfect $r$th power, the algorithm `IsPerfectRthPowerZ` always reports this. If $f$ is not a perfect $r$th power, on any invocation of the algorithm, this is reported correctly with probability at least $1 - \epsilon$.

**Proof.** If $f$ is an $r$th power then so is $f \bmod p$ for any prime $p$, and so is any $f(\alpha) \in \mathbb{F}_p$. Thus, the algorithm always reports that $f$ is an $r$th power. Now suppose $f$ is not an $r$th power. If $p \,|\, \mathrm{disc}(f)$ it may happen that $f \bmod p$ *is* an $r$th power. This happens with probability at most 1/4 and we will assume that the worst happens in this case. When $p \nmid \mathrm{disc}(f)$, the probability that `IsPerfectRthPowerGF` incorrectly reports that $f$ is an $r$th power is also at most 1/4, by our choice of parameter $\epsilon$. Thus, on any iteration of steps 4–6, the probability of finding that $f$ is an $r$th power is at most 1/2. The probability of this happening $\lceil \log_2(1/\epsilon) \rceil$ times is clearly at most $\epsilon$. □

**Theorem 2.7.** On inputs as specified, the algorithm `IsPerfectRthPowerZ` requires

$$O\Big( r\mathrm{M}(r) \log r \cdot \mathrm{M}(\log n + \log\log \|f\|_\infty) \cdot (\log n + \log\log \|f\|_\infty) \cdot \log(1/\epsilon) \Big),$$

or $O^{\sim}(r^2(\log n + \log\log \|f\|_\infty)^2 \cdot \log(1/\epsilon))$ bit operations, plus the cost to evaluate $(\alpha, p) \mapsto f(\alpha) \bmod p$ at $O(\log(1/\epsilon))$ points $\alpha \in \mathbb{F}_p$ for primes $p$ with $\log p \in O(\log n + \log\log \|f\|_\infty)$.

**Proof.** The number of operations required by each iteration is dominated by Step 5, for which $O(r\mathrm{M}(r) \log r \log p)$ operations in $\mathbb{F}_p$ is sufficient by Theorem 2.3. Since $\log p \in O(\log n + \log\log \|f\|_\infty)$ we obtain the final complexity as stated. □

We obtain the following corollary for $t$-sparse polynomials in $\mathbb{Z}[x]$. This follows since the cost of evaluating a $t$-sparse polynomial $f \in \mathbb{Z}[x]$ modulo a prime $p$ is $O(t \log \|f\|_\infty \log p + t \log n \mathrm{M}(\log p))$ bit operations.

**Corollary 2.8.** Given $f \in \mathbb{Z}[x]$ of degree $n$, with $\tau(f) = t$, and $r \in \mathbb{N}$ a prime dividing $n$, we can determine if $f$ is an $r$th power with

$$O^{\sim}\left( (r^2 \log^2 n + t \log^2 n + t \log \|f\|_\infty \log n) \cdot \log(1/\epsilon) \right)$$

bit operations. When $f$ is an $r$th power, the output is always correct, while if $f$ is not an $r$th power, the output is correct with probability at least $1 - \epsilon$.

*2.4. An upper bound on r.*

In this subsection we show that if $f = h^r$ and $f \neq x^n$ then $r$ must be small. Over $\mathbb{Z}[x]$ we show that $\|h\|_2$ is small as well. A sufficiently strong result over many fields is demonstrated by Schinzel (1987), Theorem 1, where it is shown that if $f$ has sparsity $t \geq 2$ then $t \geq r + 1$ (in fact a stronger result is shown involving the sparsity of $h$ as well). This holds when either the characteristic of the ground field of $f$ is zero or greater than $\deg f$.

Here we give a (much) simpler result for polynomials in $\mathbb{Z}[x]$, which bounds $\|h\|_2$ and is stronger at least in its dependency on $t$ though it also depends upon the coefficients of $f$.

**Theorem 2.9.** Suppose $f \in \mathbb{Z}[x]$ with $\deg f = n$ and $\tau(f) = t$, and $f = h^r$ for some $h \in \mathbb{Z}[x]$ of degree $s$ and $r \geq 2$. Then $\|h\|_2 \leq \|f\|_1^{1/r}$.

**Proof.** Let $p > n$ be prime and $\zeta \in \mathbb{C}$ a $p$th primitive root of unity. Then

$$\|h\|_2^2 = \sum_{0 \leq i \leq s} |h_i|^2 = \frac{1}{p} \sum_{0 \leq i < p} |h(\zeta^i)|^2.$$

(this follows from the fact that the Discrete Fourier Transform (DFT) matrix is orthogonal). In other words, the average value of $|h(\zeta^i)|^2$ for $i = 0 \ldots p-1$ is $\|h\|_2^2$, and so there exists a $k \in \{0, \ldots, p-1\}$ with $|h(\zeta^k)|^2 \geq \|h\|_2^2$. Let $\theta = \zeta^k$. Then clearly $|h(\theta)| \geq \|h\|_2$. We also note that $f(\theta) = h(\theta)^r$ and $|f(\theta)| \leq \|f\|_1$, since $|\theta| = 1$. Thus,

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

The following corollary is particularly useful.

**Corollary 2.10.** If $f \in \mathbb{Z}[x]$ is not of the form $x^n$, and $f = h^r$ for some $h \in \mathbb{Z}[x]$, then

(i) $r \leq 2 \log_2 \|f\|_1$.
(ii) $\tau(h) \leq \|f\|_1^{2/r}$

**Proof.** Part (i) follows since $\|h\|_2 \geq \sqrt{2}$. Part (ii) follows because $\|h\|_2 \geq \sqrt{\tau(h)}$. $\square$

These bounds relate to the sparsity of $f$ since $\|f\|_1 \leq \tau(f)\|f\|_\infty$.

*2.5. Perfect Power Detection Algorithm*

We can now complete the perfect power detection algorithm, when we are given only the $t$-sparse polynomial $f$ (and not $r$).

**Algorithm** `IsPerfectPowerZ`

**Input:** $f \in \mathbb{Z}[x]$ of degree $n$ and sparsity $t \geq 2$, $\epsilon \in \mathbb{R}_{>0}$
**Output:** True and $r$ if $f = h^r$ for some $h \in \mathbb{Z}[x]$
False otherwise.

1: $\mathcal{P} \leftarrow \{\text{primes } r \,|\, n \text{ and } r \leq 2\log_2(t\|f\|_\infty)\}$
2: **for** $r \in \mathcal{P}$ **do**
3: **if** `IsPerfectRthPowerZ`$(f, r, \epsilon/\#\mathcal{P})$ **then**
4: **return** True and $r$
5: **return** False

**Theorem 2.11.** If $f \in \mathbb{Z}[x] = h^r$ for some $h \in \mathbb{Z}[x]$, the algorithm `IsPerfectPowerZ` always returns "True" and returns $r$ correctly with probability at least $1-\epsilon$. Otherwise, it returns "False" with probability at least $1-\epsilon$. The algorithm requires $O\tilde{}(t\log^2\|f\|_\infty \cdot \log^2(n)\cdot\log(1/\epsilon))$ bit operations.

**Proof.** From the preceding discussions, we can see that if $f$ is a perfect power, then it must be a perfect $r$th power for some $r \in \mathcal{P}$. So the algorithm must return true on some iteration of the loop. However, it may incorrectly return true *too early* for an $r$ such that $f$ is not actually an $r$th power; the probability of this occurring is the probability of error when $f$ is not a perfect power, and is less than $\epsilon/\#\mathcal{P}$ at each iteration. So the probability of error on any iteration is at most $\epsilon$, which is what we wanted.

The complexity result follows from the fact that each $r \in O(\log t + \log\|f\|_\infty)$ and using Corollary 2.8. □

For polynomials in $\mathbb{F}_q[x]$ we use Schinzel's bound that $r \leq t-1$ and obtain the following algorithm.

**Algorithm** `IsPerfectPowerGF`

**Input:** $f \in \mathbb{F}_q[x]$ of degree $n$ and sparsity $t$, where the characteristic of $\mathbb{F}_q$ is greater than $n$, and $\epsilon \in \mathbb{R}_{>0}$
**Output:** True and $r$ if $f = h^r$ for some $h \in \mathbb{F}_q[x]$;
False otherwise.

1: $\mathcal{P} \leftarrow \{\text{primes } r \,|\, n \text{ and } r \leq t\}$
2: **for** $p \in \mathcal{P}$ **do**
3: **if** `IsPerfectRthPowerGF`$(f, r, \epsilon/\#\mathcal{P})$ **then**
4: **return** True and r;

**Theorem 2.12.** If $f = h^r$ for $h \in \mathbb{F}_q[x]$, the algorithm `IsPerfectPowerGF` always returns "True" and returns $r$ correctly with probability at least $1-\epsilon$. Otherwise, it returns "False" with probability at least $1-\epsilon$. The algorithm requires $O\tilde{}(t^3(\log q + \log n))$ operations in $\mathbb{F}_q$.

**Proof.** The proof is equivalent to that of Theorem 2.11, using the complexity bounds in Corollary 2.4. □

*2.6. Detecting multivariate perfect powers*

In this subsection we examine the problem of detecting multivariate perfect powers. That is, given a lacunary $f \in \mathsf{F}[x_1, \ldots, x_\ell]$ of total degree $n$ as in (1.1), we want to determine if $f = h^r$ for some $h \in \mathsf{F}[x_1, \ldots, x_\ell]$ and $r \in \mathbb{N}$. This is done simply as a reduction to the univariate case.

First, given $f \in \mathsf{F}[x_1, \ldots, x_\ell]$, define the squarefree part $\tilde{f} \in \mathsf{F}[x_1, \ldots, x_\ell]$ as the squarefree polynomial of highest total degree which divides $f$.

**Lemma 2.13.** Let $f \in \mathsf{F}[x_1, \ldots, x_\ell]$ be of total degree $n > 0$ and let $\tilde{f} \in \mathsf{F}[x_1, \ldots, x_\ell]$ be the squarefree part of $f$. Define

$$\Delta = \operatorname{disc}_x(\tilde{f}(y_1 x, \ldots, y_\ell x)) = \operatorname{res}_x(\tilde{f}(y_1 x, \ldots, y_\ell x), \tilde{f}'(y_1 x, \ldots, y_\ell x)) \in \mathsf{F}[y_1, \ldots, y_\ell]$$

and

$$\Lambda = \operatorname{lcoeff}_x(f(y_1 x, \ldots, y_\ell x)) \in \mathsf{F}[y_1, \ldots, y_\ell]$$

for independent indeterminates $x, y_1, \ldots, y_\ell$. Assume that $a_1, \ldots, a_\ell \in \mathsf{F}$ with $\Delta(a_1, \ldots, a_\ell) \neq 0$ and $\Lambda(a_1, \ldots, a_n) \neq 0$. Then $f(x_1, \ldots, x_\ell)$ is a perfect power if and only if $f(a_1 x, \ldots, a_\ell x) \in \mathsf{F}[x]$ is a perfect power.

**Proof.** Clearly if $f$ is a perfect power, then $f(a_1 x, \ldots, a_\ell x)$ is a perfect power. To prove the converse, assume that

$$f = f_1^{s_1} f_2^{s_2} \cdots f_m^{s_m}$$

for irreducible $f_1, \ldots, f_m \in \mathsf{F}[x_1, \ldots, x_\ell]$. Then

$$f(y_1 x, \ldots, y_m x) = f_1(y_1 x, \ldots, y_m x)^{s_1} \cdots f_m(y_1 x, \ldots, y_m x)^{s_m}$$

and each of the $f_i(y_1 x, \ldots, y_m x)$ are irreducible. Now, since $\Lambda(a_1, \ldots, a_m) \neq 0$, we know the $\deg(f(a_1 x, \ldots, a_\ell x)) = \deg f$ (the total degree of $f$). Thus, $\deg f_i(a_1 x, \ldots, a_\ell x) = \deg f_i$ for $1 \leq i \leq \ell$ as well. Also, by our assumption, $\operatorname{disc}(f(a_1 x, \ldots, a_\ell x)) \neq 0$, so all of the $f_i(a_1 x, \ldots, a_\ell x)$ are squarefree and pairwise relatively prime for $1 \leq i \leq k$, and

$$f(a_1 x, \ldots, a_\ell x) = f_1(a_1 x, \ldots, a_\ell x)^{s_1} \cdots f_m(a_1 x, \ldots, a_\ell x)^{s_m}.$$

Assume now that $f(a_1 x, \ldots, a_\ell x)$ is an $r$th perfect power. Then $r$ divides $s_i$ for $1 \leq i \leq m$. This immediately imples that $f$ itself is an $r$th perfect power. □

It is easy to see that the total degree of $\Delta$ is less than $2n^2$ and the total degree of $\Lambda$ is less than $n$, and that both $\Delta$ and $\Lambda$ are non-zero. Thus, for randomly chosen $a_1, \ldots, a_\ell$ from a set $\mathcal{S} \subseteq \mathsf{F}$ of size at least $8n^2 + 4n$ we have $\Delta(a_1, \ldots, a_\ell) = 0$ or $\Lambda(a_1, \ldots, a_\ell) = 0$ with probability less than $1/4$, by Zippel (1979) or Schwartz (1980). This can be made arbitrarily small by increasing the set size and/or repetition. We then run the appropriate univariate algorithm over $\mathsf{F}[x]$ (depending upon the field) to identify whether or not $f$ is a perfect power, and if so, to find $r$.

## 3. Computing perfect roots

Once we have determined that $f \in \mathsf{F}[x]$ is equal to $h^r$ for some $h \in \mathsf{F}[x]$, the next task is to actually compute $h$. Unfortunately, as noted in the introduction, there are no known bounds on $\tau(h)$ which are polynomial in $\tau(f)$.

The question of how sparse the polynomial root of a sparse polynomial must be (or equivalently, how dense any power of a dense polynomial must be) relates to some questions first raised by Erdös (1949) on the number of terms in the square of a polynomial. Schinzel extended this work to the case of perfect powers and proved that $\tau(h^r)$ tends to infinity as $\tau(h)$ tends to infinity (Schinzel, 1987). Some conjectures of Schinzel suggest that $\tau(h)$ should be $O(\tau(f))$. A recent breakthrough of Zannier (2007) show that $\tau(h)$ is bounded by a function which does not depend on $\deg f$, but this bound is unfortunately not polynomial in $\tau(f)$.

However, our own (limited) investigations, along with more extensive ones by Coppersmith and Davenport (1991), and later Abbott (2002), suggest that, for any $h \in \mathsf{F}[x]$, where the characteristic of $\mathsf{F}$ is not too small, $\tau(h) \in O(\tau(h^r) + r)$. We skirt this problem here by simply making our algorithms output sensitive; the time required is polynomial in the lacunary size of the input and the output.

### *3.1. Computing rth roots in polynomial-time (without conditions)*

In this subsection we present an algorithm for computing an $h$ such that $f = h^r$ given $f \in \mathbb{Z}[x]$ and $r \in \mathbb{Z}$ and assuming that such an $h$ exists. The algorithm requires time polynomial in $t = \tau(f)$, $\log \deg f$, $\log \|f\|_\infty$ and a given upper bound $\mu \geq m = \tau(h)$. It is not conditional on any conjectures, but is probabilistic of the Monte Carlo type. That is, the computed polynomial $h$ is such that $h^r = f$ with high probability. We will only demonstrate that this algorithm requires polynomial time. A more detailed analysis is performed on the (more efficient) algorithm of the next subsection (though that complexity is subject to a modest conjecture).

The basic idea of the algorithm here is that we can recover all the coefficients in $\mathbb{Q}$ as well as modular information about the exponents of $h$ from a homomorphism into a small cylotomic field over $\mathbb{Q}$. Doing this for a relatively small number of cyclotomic fields yields $h$.

Assume that (the unknown) $h \in \mathbb{Z}[x]$ has form

$$h = \sum_{1 \leq i \leq m} b_i x^{d_i} \quad \text{for } b_1, \ldots, b_m \in \mathbb{Z}\backslash\{0\}, \text{ and } 0 \leq d_1 < d_2 < \cdots < d_m,$$

and that $p > 2$ is a prime distinct from $r$ such that

$$p \nmid \prod_{1 \leq i < j \leq m} (d_j - d_i), \quad \text{and} \quad p \nmid \prod_{1 \leq i \leq m} (d_i + 1).$$

Let $\zeta_p \in \mathbb{C}$ be a $p$th primitive root of unity, and $\Phi_p = 1 + z + \cdots + z^{p-1} \in \mathbb{Z}[z]$ its minimal polynomial, the $p$th cyclotomic polynomial (which is irreducible in $\mathbb{Q}[z]$). Computationally we represent $\mathbb{Q}(\zeta_p)$ as $\mathbb{Q}[z]/(\Phi_p)$, with $\zeta_p \equiv z \bmod \Phi_p$. Observe that $\zeta_p^k = \zeta_p^{k \operatorname{rem} p}$ for any $k \in \mathbb{Z}$, where $k \operatorname{rem} p$ is the least non-negative residue of $k$ modulo $p$. Thus

$$h(\zeta_p) = h_p(\zeta_p) \quad \text{for} \quad h_p = \sum_{1 \leq i \leq m} b_i x^{d_i \operatorname{rem} p} \in \mathbb{Z}[x],$$

and $h_p$ is the unique representation of $h(\zeta_p)$ as a polynomial of degree less than $p-1$. By our choice of $p$, none of the exponents of $h$ are equivalent modulo $p$ and all the exponents reduced modulo $p$ are strictly less than $p-1$ (since our conditions imply $d_i \not\equiv (p-1) \bmod p$ for $1 \leq i \leq m$). This also implies that the coefficients of $h_p$ are exactly the same as those of $h$, albeit in a different order.

Now observe that we can determine $h_p$ quite easily from the roots of

$$\Gamma_p(y) = y^r - f(\zeta_p) \in \mathbb{Q}(\zeta_p)[y].$$

These roots can be found by factoring the polynomial $\Gamma_p(y)$ in $\mathbb{Q}(\zeta_p)[y]$, and the roots in $\mathbb{C}$ must be $\omega^i h(\zeta_p) \in \mathbb{C}$ for $0 \leq i < r$, where $\omega$ is a primitive $r$th root of unity. When $r > 2$, and since we chose $p$ distinct from $r$, the only $r$th root of unity in $\mathbb{Q}(\zeta_p)$ is 1. Thus $\Gamma_p(y)$ has exactly one linear factor, and this must equal to $y - h(\zeta_p) = y - h_p(\zeta_p)$, precisely determining $h_p$. When $r = 2$, we have

$$\Gamma_p(y) = (y - h(\zeta_p))(y + h(\zeta_p)) = (y - h_p(\zeta_p))(y + h_p(\zeta_p))$$

and we can only determine $h_p(\zeta_p)$ (and $h_p$ and, for that matter, $h$) up to a factor of $\pm 1$. However, the exponents of $h_p$ and $-h_p$ are the same, and the ambiguity is only in the coefficients (which we resolve later).

Finally, we need to perform the above operations for a sequence of cyclotomic fields $\mathbb{Q}(\zeta_{p_1})$, $\mathbb{Q}(\zeta_{p_2})$, $\ldots$, $\mathbb{Q}(\zeta_{p_k})$ such that the primes in $\mathcal{P} = \{p_1, \ldots, p_k\}$ allow us to recover all the exponents in $h$. Each prime $p \in \mathcal{P}$ gives the set of exponents of $h$ reduced modulo that prime, as well as *all* the coefficients of $h$ in $\mathbb{Z}$. That is, from each of the computations with $p \in \mathcal{P}$ we obtain

$$\mathcal{C} = \{b_1, \ldots, b_m\} \quad \text{and} \quad \mathcal{E}_p = \{d_1 \operatorname{rem} p, d_2 \operatorname{rem} p, \ldots, d \operatorname{rem} p\},$$

but with no clear information about the order of these sets. In particular, it is not obvious how to correlate the exponents modulo the different primes directly. To do this we employ the clever sparse interpolation technique of Garg and Schost (2008) (based on a method of Grigoriev and Karpinski (1987) for a different problem), which interpolates the symmetric polynomial in the exponents:

$$g = (x - d_1)(x - d_2) \cdots (x - d_m) \in \mathbb{Z}[x].$$

For each $p \in \mathcal{P}$ we compute the symmetric polynomial modulo $p$,

$$g_p = (x - (d_1 \operatorname{rem} p))(x - (d_2 \operatorname{rem} p)) \cdots (x - (d_m \operatorname{rem} p)) \equiv g \bmod p,$$

for which we do not need to know the order of the exponent residues. We then determine $g \in \mathbb{Z}[x]$ by the Chinese remainder theorem and factor $g$ over $\mathbb{Z}[x]$ to find the $d_1, \ldots, d_m \in \mathbb{Z}$. Thus the product of all primes in $p \in \mathcal{P}$ must be at least $2\|g\|_\infty$ to recover the coefficients of $g$ uniquely. It is easily seen that $2\|g\|_\infty \leq 2n^m$.

As noted above, the computation with each $p \in \mathcal{P}$ recovers all the exponents of $h$ in $\mathbb{Z}$, so using only one prime $p \in \mathcal{P}$, we determine the $j$th exponent of $h$ as the coefficient of $x^{d_j \operatorname{rem} p}$ in $h_p$ for $1 \leq j \leq m$. If $r = 2$ we can choose either of the roots of $\Gamma_p(y)$ (they differ by only a sign) to recover the coefficients of $h$.

The above discussion is summarized in the following algorithm.

**Algorithm** `ComputeRootAlgebraic`

**Input:** $f \in \mathbb{Z}[x]$ as in (1.2) and $r, \mu \in \mathbb{N}$.
**Output:** $h \in \mathbb{Z}[x]$ such that $f = h^r$ and $\tau(h) \le \mu$, provided such an $h$ exists.

1: $\gamma \leftarrow$ smallest integer such that $2\gamma/5 \log \gamma \ge 10\mu^2(\log_2 n)(1 + \mu \log_2 n)$.
2: $\mathcal{P} \leftarrow$ set of $k > m \log_2 n$ primes chosen uniformly at random from $\{\gamma, \ldots, 2\gamma\}$.
3: **for** $p \in \mathcal{P}$ **do**
4: Represent $\mathbb{Q}(\zeta_p)$ by $\mathbb{Q}[x]/(\Phi_p)$, where $\Phi_p = 1 + z + \cdots + z^{p-1}$ and $\zeta_p \equiv z \bmod \Phi_p$.
5: Compute $f(\zeta_p) = \sum_{1 \le i \le t} c_i \zeta_p^{e_i \bmod p} \in \mathbb{Q}(\zeta_p)$.
6: $h_p \leftarrow$ root of $\Gamma_p = y^r - f(\zeta_p)$ in $\mathbb{Q}(\zeta_p)$, found by factoring $\Gamma_p$ over $\mathbb{Q}(\zeta_p)[y]$.
7: **if** $\deg h_p \ge p - 1$ or $h_p$ has non-integer coefficients **then**
8: **return** `FAIL`
9: Write $h_p \in \mathbb{Z}[x]$ as $\sum_{1 \le i \le m} b_{ip} x^{d_{ip}}$.
10: **if** $m$ differs from previous values of $m$ **then**
11: **return** `FAIL`
12: $g_p \leftarrow (x - d_{1p})(x - d_{2p}) \cdots (x - d_{mp}) \in \mathbb{Z}_p[x]$.
13: Reconstruct $g \in \mathbb{Z}[x]$ from $\{g_p\}_{p \in \mathcal{P}}$ by the Chinese remainder algorithm.
14: $\{d_1, d_2, \ldots, d_m\} \leftarrow$ distinct integer roots of $g \in \mathbb{Z}[x]$.
15: Choose any $p \in \mathcal{P}$. For $1 \le j \le m$, let $b_j \in \mathbb{Z}$ be the coefficient of $x^{d_j \bmod p}$ in $h_p$.
16: Return $h = \sum_{1 \le i \le m} b_j x^{d_j}$.

**Theorem 3.1.** The algorithm `ComputeRootAlgebraic` works as stated. It is probabilistic of the Monte Carlo type and returns the correct answer with probability at least 9/10 on any execution. It requires a number of bit operations polynomial in $t = \tau(f)$, $\log \deg f$, $\log \|f\|$, and $\mu$.

**Proof.** In Step 1 we need to choose a set of primes $\mathcal{P}$ which are all *good* with sufficiently high probability, in the sense that for all $p \in \mathcal{P}$

$$\beta = r \cdot \prod_{1 \le i < j \le m} (d_j - d_i) \cdot \prod_{1 \le i \le m} (d_i + 1) \not\equiv 0 \bmod p.$$

It is easily derived that $\beta \le n^{\mu^2}$, which has fewer than $\log_2 \beta \le \mu^2 \log_2 n$ prime factors. We also need to recover $g$ in Step 7, and $\|g\|_\infty \le n^\mu$, so we need at least $1 + \log_2 \|g\| \le 1 + \mu \log_2 n$ primes. Thus, if $\mathcal{P}$ has at least $10\mu^2 \log_2(n)(1 + \mu \log_2 n)$, the probability of choosing a bad prime from $\mathcal{P}$ is at most $1/(10(1 + \mu \log_2 n))$. The probability of choosing a bad prime with $(1 + \mu \log_2 n)$ choices is at most 1/10, and the probability that all the primes are good is at least 9/10. Numbers are chosen uniformly and randomly from $\{\gamma, \ldots, 2\gamma\}$ and tested for primality, say by Agrawal et al. (2004). Correctness of the remainder of the algorithm follows from the previous discussion. Factoring the polynomials $\Gamma_p \in \mathbb{Q}(\zeta_p)[y]$ can be performed in polynomial time with the algorithm of, for example, Landau (1985), and all other steps clearly require polynomial time as well. □

### *3.2. Faster root computation subject to conjecture*

Algorithm `ComputeRootAlgebraic` is probabilistic of the Monte Carlo type and not of the Las Vegas type because we have no way of certifying the output — i.e. that $h^r = f$ for given lacunary $h, f \in \mathbb{Z}[x]$ — in polynomial time. One way to accomplish this would

be to simply compute $h^r$ by repeated squaring and comparing the result to $f$, but to do so in polynomial time would require bounds on the sparsity of each intermediate power $\tau(h^i)$ for $2 \leq i < r$ based on $\tau(h)$ and $\tau(f)$.

In fact, with such sparsity bounds we can actually derive a deterministic algorithm based on Newton iteration. This approach does not rely on advanced techniques such as factoring over algebraic extension fields, and hence will be much more efficient in practice. It is also more general as it applies to fields other than $\mathbb{Z}$ and to powers $r$ which are not prime.

Unfortunately, this algorithm is not purely output-sensitive, as it relies on the following conjecture regarding the sparsity of powers of $h$:

**Conjecture 3.2.** For $r, s \in \mathbb{N}$, if the characteristic of $\mathsf{F}$ is zero or greater than $rs$, and $h \in \mathsf{F}[x]$ with $\deg h = s$, then

$$\tau(h^i \bmod x^{2s}) < \tau(h^r \bmod x^{2s}) + r, \qquad i = 1, 2, \ldots, r-1.$$

This corresponds to intuition and experience, as the system is still overly contrained with only $s$ degrees of freedom. A weaker conjecture would suffice to prove polynomial time, but we use the stated bounds as we believe these give more accurate complexity measures.

Our algorithm is essentially a Newton iteration, with special care taken to preserve sparsity. We start with the image of $h$ modulo $x$, using the fact that $f(0) = h(0)^r$, and at Step $i = 1, 2, \ldots, \lceil \log_2(\deg h + 1) \rceil$, we compute the image of $h$ modulo $x^i$.

Here, and for the remainder of this section, we will assume that $f, h \in \mathsf{F}[x]$ with degrees $n$ and $s$ respectively such that $f = h^r$ for $r \in \mathbb{N}$ at least 2, and that the characteristic of $\mathsf{F}$ is either zero or greater than $n$. As usual, we define $t = \tau(f)$. We require the following simple lemma.

**Lemma 3.3.*** Let $k, \ell \in \mathbb{N}$ such that $\ell \leq k$ and $k + \ell \leq s$, and suppose $h_1 \in \mathsf{F}[x]$ is the unique polynomial with degree less than $k$ satisfying $h_1^r \equiv f \bmod x^k$. Then

$$\tau(h_l^{r+1} \bmod x^{k+\ell}) \leq 2t(t+r).$$

**Proof.** Let $h_2 \in \mathsf{F}[x]$ be the unique polynomial of degree less than $\ell$ satisfying $h_1 + h_2 x^k \equiv h \bmod x^{k+\ell}$. Since $h^r = f$,

$$f \equiv h_1^r + r h_1^{r-1} h_2 x^k \mod x^{k+\ell}.$$

Multiplying by $h_1$ and rearranging gives

$$h_1^{r+1} \equiv h_1 f - r f h_2 x^k \mod x^{k+\ell}.$$

Because $h_1 \bmod x^k$ and $h_2 \bmod x^\ell$ each have at most $\tau(h)$ terms, which by Conjecture 3.2 is less than $t - r$, the total number of terms in $h_1^{r-1} \bmod x^{k+\ell}$ is less than $2t(t-r)$. □

This essentially tells us that the "error" introduced by examining higher-order terms of $h_1^r$ is not too dense. It leads to the following algorithm for computing $h$.

* Lemma subject to the validity of Conjecture 3.2.

**Algorithm** `ComputeRootNewton`

**Input:** $f \in \mathsf{F}[x]$, $r \in \mathbb{N}$ such that $f$ is a perfect $r$th power
**Output:** $h \in \mathsf{F}[x]$ such that $f = h^r$

1: $u \leftarrow$ highest power of $x$ dividing $f$
2: $f_u \leftarrow$ coefficient of $x^u$ in $f$
3: $g \leftarrow f/(f_u x^u)$
4: $h \leftarrow 1, \quad k \leftarrow 1$
5: **while** $kr \leq \deg g$ **do**
6: $\quad \ell \leftarrow \min\{k, (\deg g)/r + 1 - k\}$
7: $\quad a \leftarrow \dfrac{hg - h^{r+1} \bmod x^{k+\ell}}{rx^k}$
8: $\quad h \leftarrow h + (a/g \bmod x^\ell)x^k$
9: $\quad k \leftarrow k + \ell$
10: $b \leftarrow$ *any* $r$th root of $f_u$ in $\mathsf{F}$
11: **return** $bhx^{u/r}$

**Theorem 3.4.** If $f \in \mathsf{F}[x]$ is a perfect $r$th power, then `ComputeRootNewton` returns an $h \in \mathsf{F}[x]$ such that $h^r = f$.

**Proof.** Let $u, f_u, g$ be as defined in Steps 1–4. Thus $f = f_u g x^u$. Now let $\hat{h}$ be some $r$th root of $f$, which we assume exists. If we similarly write $\hat{h} = \hat{h}_v \hat{g} x^v$, with $\hat{h}_v \in \mathsf{F}$ and $\hat{g} \in \mathsf{F}[x]$ such that $\hat{g}(0) = 1$, then $\hat{h}^r = {\hat{h}_v}^r \hat{g}^r x^{vr}$. Therefore $f_u$ must be a perfect $r$th power in $\mathsf{F}$, $r|u$, and $g$ is a perfect $r$th power in $\mathsf{F}[x]$ of some polynomial with constant coefficient equal to 1.

Denote by $h_i$ the value of $h$ at the beginning of the $i$th iteration of the while loop. So $h_1 = 1$. We claim that at each iteration through Step 6, $h_i^r \equiv g \bmod x^k$. From the discussion above, this holds for $i = 1$. Assuming the claim holds for all $i = 1, 2, \ldots, j$, we prove it also holds for $i = j + 1$.

From Step 8, $h_{j+1} = h_j + (a/g \bmod x^l)x^k$, where $a$ is as defined on the $j$th iteration of Step 7. We observe that

$$h_j h_j^r \equiv h_j^{r+1} + r h_j^r (a/g \bmod x^l) x^k \mod x^{k+\ell}.$$

From our assumption, $h_j^r \equiv f \bmod x^k$, and $l \leq k$, so we have

$$h_j h_{j+1}^r \equiv h_j^{r+1} + rax^k \equiv h_j^{r+1} + h_j f - h_j^{r+1} \equiv h_j f \mod x^{k+\ell}$$

Therefore $h_{j+1}^r \equiv f \bmod x^{k+\ell}$, and so by induction the claim holds at each step. Since the algorithm terminates when $kr > \deg g$, we can see that the final value of $h$ is an $r$th root of $g$. Finally, $\left(bhx^{u/r}\right)^r = f_u g x^u = f$, so the theorem holds. □

**Theorem 3.5.**[†] If $f \in \mathsf{F}[x]$ has degree $n$ and $t$ nonzero terms, then `ComputeRootNewton` uses $O\left((t+r)^4 \log r \log n\right)$ operations in $\mathsf{F}$ and an additional $O\left((t+r)^4 \log r \log^2 n\right)$ bit operations, not counting the cost of root-finding in the base field $\mathsf{F}$ on Step 10.

[†] Theorem subject to the validity of Conjecture 3.2.

**Proof.** First consider the cost of computing $h^{r+1}$ in Step 7. This will be accomplished by repeatedly squaring and multiplying by $h$, for a total of at most $2\lfloor \log_2(r+1) \rfloor$ multiplications. As well, each intermediate product will have at most $\tau(f) + r < (t+r)^2$ terms, by Conjecture 3.2. The number of field operations required, at each iteration, is $O\left((t+r)^4 \log r\right)$, for a total cost of $O\left((t+r)^4 \log r \log n\right)$.

Furthermore, since $k + \ell \leq 2^i$ at the $i$'th step, for $1 \leq i < \log_2 n$, the total cost in bit operations is less than

$$\sum_{1 \leq i < \log_2 n} (t+r)^4 \log_2 ri \in O\left((t+r)^4 \log r \log^2 n\right).$$

In fact, this is the most costly step. The initialization in Steps 1–4 uses only $O(t)$ operations in $\mathsf{F}$ and on integers at most $n$. And the cost of computing the quotient on Step 8 is proportional to the cost of multiplying the quotient and dividend, which is at most $O(t(t+r))$. □

When $\mathsf{F} = \mathbb{Q}$, we must account for coefficient growth. We use the normal notion of the size of a rational number: For $\alpha \in \mathbb{Q}$, write $\alpha = a/b$ for $a, b$ relatively prime integers. Then define $\mathcal{H}(\alpha) = \max\{|a|, |b|\}$. And for $f \in \mathbb{Q}[x]$ with coefficients $c_1, \ldots, c_t \in \mathbb{Q}$, write $\mathcal{H}(f) = \max \mathcal{H}(c_i)$.

Thus, the size of the lacunary representation of $f \in \mathbb{Q}[x]$ is proportional to $\tau(f)$, $\deg f$, and $\log \mathcal{H}(f)$. Now we prove the bit complexity of our algorithm is polynomial in these values, when $\mathsf{F} = \mathbb{Q}$.

**Theorem 3.6.**† Suppose $f \in \mathbb{Q}[x]$ has degree $n$ and $t$ nonzero terms, and is a perfect $r$th power. `ComputeRootNewton` computes an $r$th root of $f$ using $O^{\sim}\left(t(t+r)^4 \cdot \log n \cdot \log \mathcal{H}(f)\right)$ bit operations.

**Proof.** Let $h \in \mathbb{Q}[x]$ such that $h^r = f$, and let $c \in \mathbb{Z}_{>0}$ be minimal such that $ch \in \mathbb{Z}[x]$. Gauß's Lemma tells us that $c^r$ must be the least positive integer such that $c^r f \in \mathbb{Z}[x]$ as well. Then, using Theorem 2.9, we have:

$$\mathcal{H}(h) \leq \|ch\|_\infty \leq \|ch\|_2 \leq (t\|c^r f\|_\infty)^{1/r} \leq t^{1/r} \mathcal{H}(f)^{(t+1)/r}.$$

(The last inequality comes from the fact that the lcm of the denominators of $f$ is at most $\mathcal{H}(f)^t$.)

Hence $\log \mathcal{H}(h) \in O\left((t \log \mathcal{H}(f))/r\right)$. Clearly the most costly step in the algorithm will still be the computation of $h_i^{r+1}$ at each iteration through Step 7. For simplicity in our analysis, we can just treat $h_i$ (the value of $h$ at the $i$th iteration of the while loop in our algorithm) as equal to $h$ (the *actual* root of $f$), since we know $\tau(h_i) \leq \tau(h)$ and $\mathcal{H}(h_i) \leq \mathcal{H}(h)$.

Lemma 3.3 and Conjecture 3.2 tell us that $\tau(h^i) \leq 2(t+r)^2$ for $i = 1, 2, \ldots, r$. To compute $h^{r+1}$, we will actually compute $(ch)^{r+1} \in \mathbb{Z}[x]$ by repeatedly squaring and multiplying by $ch$, and then divide out $c^{r+1}$. This requires at most $\lfloor \log_2 r + 1 \rfloor$ squares and products.

Note that $\|(ch)^{2i}\|_\infty \leq (t+r)^2 \|(ch)^i\|_\infty^2$ and $\|(ch)^{i+1}\|_\infty \leq (t+r)^2 \|(ch)^i\|_\infty \|ch\|_\infty$. Therefore

$$\|(ch)^i\|_\infty \leq (t+r)^{2r} \|ch\|_\infty^r, \quad i = 1, 2, \ldots, r,$$

and thus $\log \|(ch)^i\|_\infty \in O\left(r(t+r) + t \log \mathcal{H}(f)\right)$, for each intermediate power $(ch)^i$.

Thus each of the $O\left((t+r)^4 \log r\right)$ field operations at each iteration costs $O(\mathrm{M}(t \log \mathcal{H}(f) + \log r(t+r)))$ bit operations, which then gives the stated result. $\square$

The method used for Step 10 depends on the field $\mathsf{F}$. For $\mathsf{F} = \mathbb{Q}$, we just need to find two integer perfect roots, which can be done in "nearly linear" time by the algorithm of Bernstein (1998). Otherwise, we can use any of the well-known fast root-finding methods over $\mathsf{F}[x]$ to compute a root of $x^r - f_u$.

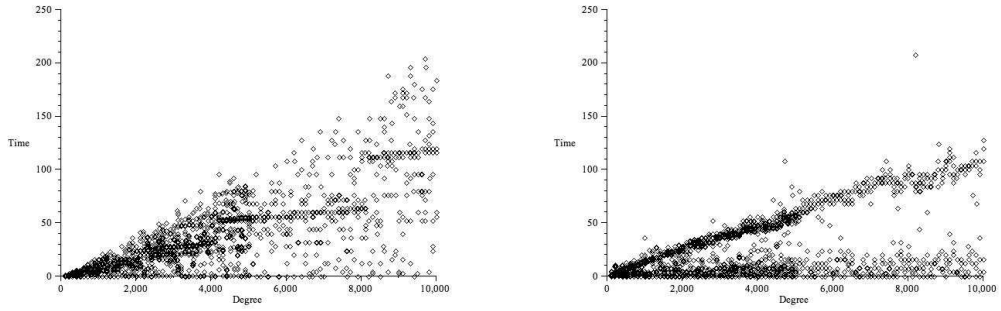


Fig. 1. Comparison of Newton Iteration (left) vs. our `IsPerfectPowerZ` (right). Inputs are dense.

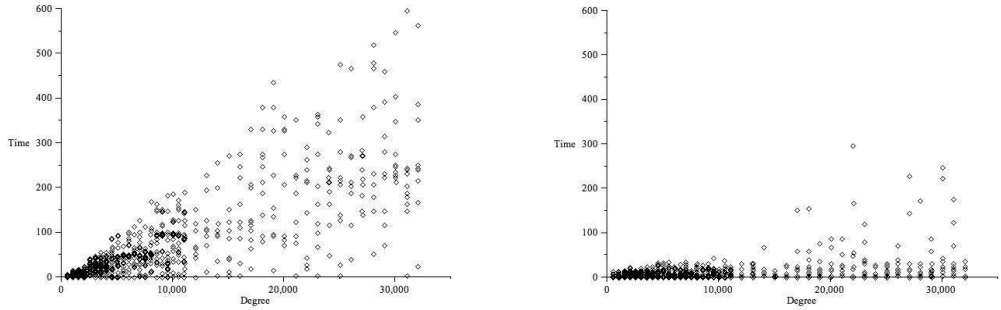


Fig. 2. Comparison of Newton Iteration (left) vs our `IsPerfectPowerZ` (right). Inputs are sparse, with sparsity fixed around 500.

## 4. Implementation

To investigate the practicality of our algorithms, we implemented `IsPerfectPowerZ` using Victor Shoup's NTL. This is a high-performance C++ for fast dense univariate polynomial computations over $\mathbb{Z}[x]$ or $\mathbb{F}_q[x]$.

NTL does not natively support a lacunary polynomial representation, so we wrote our own using vectors of coefficients and of exponents. In fact, since `IsPerfectPowerZ` is a black-box algorithm, the only sparse polynomial arithmetic we needed to implement was for evaluation at a given point.

The only significant diversion between our implementation and the algorithm specified in Section 2 is our choice of the ground field. Rather than working in a degree-$(r-1)$ extension of $\mathbb{F}_p$, we simply find a random $p$ in the same range such that $(r-1) \mid p$. It

is more difficult to prove that we can find such a $p$ quickly (using e.g. the best known bounds on Linnik's Constant), but in practice this approach is very fast because it avoids computing in field extensions.

As a point of comparison, we also implemented the Newton iteration approach to computing perfect polynomial roots, which appears to be the fastest known method for dense polynomials. This is not too dissimilar from the techniques from the previous section on computing a lacunary $r$th root, but without paying special attention to sparsity. We work modulo a randomly chosen prime $p$ to compute an $r$th perfect root $h$, and then use random evaluations of $h$ and the original input polynomial $f$ to certify correctness. This yields a Monte Carlo algorithm with the same success probability as ours, and so provides a suitable and fair comparison.

We ran two sets of tests comparing these algorithms. The first set, depicted in Figure 1, does not take advantage of sparsity at all; that is, the polynomials are dense and have close to the maximal number of terms. It appears that the worst-case running time of our algorithm is actually a bit better than the Newton iteration method on dense input, but on the average they perform roughly the same. The lower triangular shape comes from the fact that both algorithms can (and often do) terminate early. The visual gap in the timings for the sparse algorithm comes from the fact that exactly half of the input polynomials were perfect powers. It appears our algorithm terminates more quickly when the polynomial is not a perfect power, but usually takes close to the full amount of time otherwise.

The second set of tests, depicted in Figure 2, held the number of terms of the perfect power, $\tau(f)$, roughly fixed, letting the degree $n$ grow linearly. Here we can see that, for sufficiently sparse $f$, our algorithm performs significantly and consistently better than the Newton iteration. In fact, we can see that, with some notable but rare exceptions, it appears that the running time of our algorithm is largely independent of the degree when the number of terms remains fixed. The outliers we see probably come from inputs that were unluckily dense (it is not trivial to produce examples of $h^r$ with a given fixed number of nonzero terms, so the sparsity did vary to some extent).

Perhaps most surprisingly, although the choices of parameters for these two algorithms only guaranteed a probability of success of at least $1/2$, in fact over literally millions of tests performed with both algorithms and a wide range of input polynomials, not a single failure was recorded. This is of course due to the loose bounds employed in our analysis, indicating a lack of understanding at some level, but it also hints at the possibility of a deterministic algorithm, or at least one which is probabilistic of the Las Vegas type.

Both implementations are available as C++ code downloadable from the second author's website.

**Acknowledgement**

The authors would like to thank Erich Kaltofen and Igor Shparlinski for their comments.

## References


J. Abbott. Sparse squares of polynomials. *Math. Comp.*, 71(237):407–413 (electronic), 2002. ISSN 0025-5718.

M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2): 781–793, 2004.

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

M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proc. STOC 1988*, pages 301–309, New York, N.Y., 1988. ACM Press.

D. J. Bernstein. Detecting perfect powers in essentially linear time. *Mathematics of Computation*, 67(223):1253–1283, 1998.

D. Cantor and E. Kaltofen. Fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28:693–701, 1991.

D. Coppersmith and J. Davenport. Polynomials whose powers are sparse. *Acta Arith.*, 58(1):79–87, 1991. ISSN 0065-1036.

F. Cucker, P. Koiran, and S. Smale. A polynomial time algorithm for Diophantine equations in one variable. *J. Symbolic Comput.*, 27(1):21–29, 1999. ISSN 0747-7171.

P. Erdös. On the number of terms of the square of a polynomial. *Nieuw Arch. Wiskunde (2)*, 23:63–65, 1949.

S. Garg and E. Schost. Interpolation of polynomials given by straight-line programs. Preprint, 2008.

J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, New York, Melbourne, 2003. ISBN 0521826462.

J. von zur Gathen, M. Karpinski, and I. Shparlinski. Counting curves and their projections. In *ACM Symposium on Theory of Computing*, pages 805–812, 1993.

M. Giesbrecht and D. S. Roche. On lacunary polynomial perfect powers. In *ISSAC'08: Proc. International Symposium on Symbolic and Algebraic Computation*, pages 103–110. ACM, 2008.

D. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Foundations of Computer Science (FOCS)*, pages 166–172, 1987.

E. Kaltofen. Single-factor hensel lifting and its application to the straight-line complexity of certain polynomials. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 443–452, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/28395.28443.

E. Kaltofen and P. Koiran. Finding small degree factors of multivariate supersparse (lacunary) polynomials over algebraic number fields. In *ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 162–168. ACM Press, New York, NY, USA, 2006. ISBN 1-59593-276-3. doi: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/1145768.1145798.

E. Kaltofen and W-s. Lee. Early termination in sparse interpolation algorithms. *J. Symbolic Comput.*, 36(3-4):365–400, 2003. ISSN 0747-7171. International Symposium on Symbolic and Algebraic Computation (ISSAC'2002) (Lille).

M. Karpinski and I. Shparlinski. On the computational hardness of testing squarefreeness of sparse polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(027), 1999.

S. Landau. Factoring polynomials over algebraic number fields. *SIAM J. Comput.*, 14: 184–195, 1985.

H. W. Lenstra, Jr. Finding small degree factors of lacunary polynomials. In *Number theory in progress, Vol. 1 (Zakopane-Kościelisko, 1997)*, pages 267–276. de Gruyter, Berlin, 1999.

R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading MA, 1983.

M. Mignotte. An inequality about factors of polynomials. *Math. Comp.*, 28:1153–1157, 1974.

D. A. Plaisted. Sparse complex polynomials and polynomial reducibility. *J. Comp. and System Sciences*, 14:210–221, 1977.

D. A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theor. Computer Science*, 31:125–138, 1984.

A. Quick. Some gcd and divisibility problems for sparse polynomials. Technical Report 191/86, University of Toronto, 1986.

J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.

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

J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Computing Machinery*, 27:701–717, 1980.

V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, 1994.

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

A. Weil. On some exponential sums. *Proc Nat. Acad. Sci. U.S.A.*, 34:204–207, 1948.

D. Y.Y. Yun. On square-free decomposition algorithms. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 26–35, New York, NY, USA, 1976. ACM. doi: http://doi.acm.org.proxy.lib.uwaterloo.ca/10.1145/800205.806320.

U. Zannier. On the number of terms of a composite polynomial. *Acta Arith.*, 127(2): 157–167, 2007. ISSN 0065-1036.

R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. EUROSAM 79*, pages 216–226, Marseille, 1979.