> The essence of being human is that one does not seek perfection,... that one is prepared in the end to be defeated and broken up by life, which is the inevitable price of fastening one's love upon other human individuals. No doubt alcohol, tobacco, and so forth, are things that a saint must avoid, but sainthood is also a thing that human beings must avoid.
>
> —George Orwell, "Reflections on Gandhi" (1949)

Chapter 6

Sparse perfect powers

We examine the problem of detecting when a given sparse polynomial f is equal to h^r for some other polynomial h and integer $r \geq 2$. In this case we say f is a *perfect power*, and h is its rth root. We give randomized algorithms to detect when f is a perfect power, by repeatedly evaluating the polynomial in specially-chosen finite fields and checking whether the evaluations are themselves perfect powers. These detection algorithms are randomized of the Monte Carlo type, meaning that they are always fast but may return an incorrect answer (with small probability). In fact, the only kind of wrong answer ever returned will be a false positive, so we have shown that the perfect power detection problem is in the complexity class **coRP**. As a by-product, these decision problems also compute an integer $r > 2$ such that f is an rth perfect power, if such an r exists.

Once the power r is known, we turn to the problem of actually computing the perfect rth root of f, and give two different algorithms. In the case of rational number coefficients, we exhibit a deterministic algorithm that is *output-sensitive*; i.e., the cost of the algorithm depends on the size of the perfect root h in the sparse representation. The algorithm relies on polynomial factorization over algebraic extension fields to achieve output-sensitive polynomial time.

The second algorithm we consider for computing the perfect root is inspired by the Newton iteration algorithm that is most efficient for dense polynomials. Our modified Newton iteration is more sensitive to the sparsity of the problem, and carefully avoids blow-up of intermediate results, but only if a certain conjecture regarding the sparsity of intermediate powers is true. We have evidence to believe this conjecture, and under that assumption our algorithm is very efficient and practical.

A preliminary version of this work appeared at ISSAC 2008 (Giesbrecht and Roche, 2008), and further progress has been presented in an article accepted to appear in the Journal of Symbolic Computation (Giesbrecht and Roche, 2011). We are very grateful to the helpful discussions of these topics with Erich Kaltofen and Igor Shparlinski, and particularly to Pascal Koiran for pointing out the perfect power certification idea using logarithmic derivatives that we will discuss in Section 6.3.1.

6.1 Background

This chapter and its sequel focus on computational problems where the input polynomial is given in the sparse representation. Recall that this means we write

$$f = \sum_{1 \le i \le t} c_i \overline{x}^{\overline{e}_i} \in \mathsf{R}[x_1, \ldots, x_\ell], \tag{6.1}$$

where $c_1, \ldots, c_t \in \mathsf{R}^*$, $\overline{e}_1, \ldots, \overline{e}_t \in \mathbb{N}^\ell$ are distinct exponent tuples with $0 \le \|\overline{e}_1\|_1 \le \cdots \le \|\overline{e}_t\|_1 = \deg f$, and $\overline{x}^{\overline{e}_i}$ is the monomial $x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_\ell^{e_{i\ell}}$ of total degree $\|\overline{e}_i\|_1 = \sum_{1 \le j \le \ell} e_{ij}$. We say f is *t-sparse* and write $\tau(f) = t$. We present algorithms which require time polynomial in $\tau(f)$ and $\log \deg f$.

6.1.1 Related work

Computational work on sparse polynomials has proceeded steadily for the past three decades. We briefly pause to examine some of the most significant results in this area.

David Plaisted (1977; 1984) initiated the study of computational complexity for sparse polynomial computations. His early work showed, surprisingly, that some basic problems that admit fast algorithms for densely-represented polynomials are **NP**-hard when the input is sparse. In particular, Plaisted gives a reduction from 3-SAT to *relative primality testing*, the problem of determining whether a pair of sparse polynomials $f, g \in \mathbb{Z}[x]$ has any non-trivial common factor. This implies for instance that a polynomial-time greatest common divisor algorithm for sparse polynomials is unlikely to exist.

A number of other researchers have investigated complexity-theoretic problems relating to sparse polynomials as well. Recall that #**P** is the class of problems that counts accepting inputs for problems in **NP**, and important examples of a #**P**-complete problems are counting the number of satisfying assignments for a boolean formula or computing the permanent of a matrix. Quick (1986) and von zur Gathen, Karpinski, and Shparlinski (1996) followed Plaisted by proving (among other results) that the sparse gcd problem is actually #**P**-complete, when the problem is defined to determine the degree of the greatest common divisor of two given sparse polynomials.

Observe that relative primality of two polynomials corresponds to their gcd having minimal degree, while *divisibility* of sparse polynomials corresponds to the degree of their gcd being maximal. It is not yet known whether sparse divisibility testing can be performed in polynomial time, but Grigoriev, Karpinski, and Odlyzko (1996) showed the existence of short

proofs of non-divisibility, meaning the problem is in **coNP**. Later, Karpinski and Shparlinski (1999) showed that testing *square-freeness* of sparse polynomials is also **NP**-hard, via a reduction from relative primality testing.

But not every computational problem with sparse polynomials is hard. In fact, there has been significant progress in developing polynomial-time algorithms for sparse polynomial computation over the years. While some operations, such as computing derivatives, are trivially polynomial-time in the sparse representation, many problems require non-trivial algorithms. One major area of research in this direction regards sparse polynomial interpolation, the topic of the next two chapters.

Computing low-degree factors and in particular roots of sparse polynomials is another area of rapid progress in algorithmic development (Cucker, Koiran, and Smale, 1999; Lenstra, 1999; Kaltofen and Koiran, 2005, 2006). These algorithms generally take a degree bound d and a sparse polynomial f and compute all factors of f with degree at most d, in polynomial-time in d and the sparse representation size of f. Since the number of terms in high-degree factors may be exponentially larger than the size of the input, this is in some sense the best that can be hoped for without considering output-sensitive algorithms.

We now turn to the topic of this chapter, namely detecting and computing perfect power factorizations of sparse polynomials. The algorithm we present for perfect powers is interesting in that, unlike the factorization methods mentioned above, the sparse factors it produces may have exponentially high degree in the size of the input.

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 given in the dense representation. 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 at least linear time in the *degree* of f, which may be exponential in the sparse representation 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. Our algorithms, which require input in the sparse representation, are not as general, but they do avoid the dependence on the degree of h. Interestingly, we will show that the case of large r, which is important for straight-line programs, cannot happen for sparse polynomials, at least over the rings that we will consider.

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; our Algorithm 6.4 demonstrates that this problem can be solved in randomized polynomial-time.

6.1.2 Overview of results

We will always assume that the sparsity $\tau(f) \geq 2$. Otherwise $f = x^n$, and determining whether f is a perfect power is equivalent to determining whether $n \in \mathbb{N}$ is composite. This is of course tractable, but producing an $r \geq 2$ such that f is a perfect rth power is then equivalent to long integer factorization, a notoriously difficult problem in number theory that we do not solve here. Surprisingly, the intractability of computing such an r is avoided when $\tau(f) \geq 2$.

We first consider the problem of detecting perfect powers and computing the power r for the univariate case, where we write

$$f = \sum_{1 \leq i \leq t} c_i x^{e_i} \in \mathsf{R}[x], \tag{6.2}$$

with integer exponents $0 \leq e_1 < e_2 < \cdots < e_t = \deg f$.

Two cases for the ring R are handled: the integers and finite fields of characteristic p greater than the degree of f. When f has integer coefficients, our algorithms also require time polynomial in the size of those integers in the IMM model.

To be precise about this definition of size, first recall from Chapter 1 that size(a) for $a \in \mathbb{Z}$ is the number of machine words needed to represent a in an IMM. So, if w is the size of machine words, size$(a) = \lceil \log_2(|a|+1)/w \rceil + 1$. For $f \in \mathbb{Z}[x]$ written as in (6.2), define:

$$\text{size}(f) = \sum_{i=1}^{t} \text{size}(c_i) + \sum_{i=1}^{t} \text{size}(e_i). \tag{6.3}$$

We will often employ the following upper bound for simplicity:

$$\text{size}(f) \leq t \left(H(f) + \text{size}(\deg f) \right), \tag{6.4}$$

where $H(f)$ is defined as $\max_{1 \leq i \leq t} \text{size}(c_i)$.

For the analysis, as in the previous chapter, we will write $\mathsf{M}(r)$ for the number of ring operations required for degree-r dense polynomial multiplication. For $a \in \mathbb{N}$, We also define $\mathsf{N}(a)$ to be the number of IMM operations required to multiply two integers at most a in absolute value. From Section 1.4, we know that $\mathsf{N}(a) \in O(\text{size}(a) \log \text{size}(a))$. Our motivation for the $\mathsf{N}(a)$ notation is purely for clarity and brevity, not to reflect the possibility of improved algorithms for this problem in the future. One consequence is that, if the finite field elements are represented in machine words on a normal IMM, multiplication of degree-r dense polynomials in $\mathbb{F}_p[x]$ is performed with

$$O(N(p^r)) \in O(r \,\text{size}(p) \cdot \log(r + \text{size}(p))) \in O(r \mathsf{N}(p) \log r)$$

IMM operations, by encoding the whole polynomial into an integer, as discussed in Chapter 1.

Notice that the algorithm for integer polynomials will also cover those with rational coefficients, since if $f \in \mathbb{Q}[x]$, we can simply work with $\overline{f} = cf \in \mathbb{Z}[x]$ for the smallest possible $c \in \mathbb{Z}^*$.

We present polynomial-time algorithms for polynomials over finite fields and over the integers. Efficient techniques will also be presented for reducing the multivariate case to the univariate one, and for computing a root h such that $f = h^r$.

6.2 Testing for perfect powers

In this section we describe a method to determine if a sparse polynomial $f \in \mathsf{R}[x]$ is a perfect power. That is, do there exist $h \in \mathsf{R}[x]$ and $r > 1$ such that $f = h^r$? Importantly, the cost does not depend on the sparsity of h. That is, the root h could have exponentially more nonzero coefficients than the input polynomial f. However, some widely-believed conjectures indicate that this will never happen, as we will discuss later. In any case, our algorithms in this section will only compute the power $r \in \mathbb{N}$, not the polynomial root h.

We first describe algorithms to test if an $f \in \mathsf{R}[x]$ is an rth power of some polynomial $h \in \mathsf{R}[x]$, where f and r *are both given* and r is assumed to be prime. We present and analyse 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 very useful property of sparse polynomials over finite fields.

For a sparse polynomial f we then show that, in fact, if h exists at all then r must be small unless $f = x^n$. (By "small", we mean bounded in value by the size of the sparse representation of f.) If f is a non-trivial perfect power, then there certainly exists a prime r such that f is an rth power. So in fact the restrictions that r is small and prime are sufficient to cover all interesting cases, and our method is complete.

6.2.1 Detecting given rth powers

Our main tool in this work is the following theorem which says that, with reasonable probability, a polynomial is an rth power if and only if the modular image of an evaluation in a specially constructed finite field is an rth power. Put another way, if f is not an rth power, then the finite field will have sufficiently many witnesses to this fact.

Theorem 6.1. *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 rth power in $\mathbb{F}_\varrho[x]$. Then*

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

Proof. The rth 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 rth 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 χ 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 χ 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

$$R_f^{(r)} = \sum_{a\in\mathbb{F}_\varrho^*}\left(\frac{1}{r}\sum_{\chi^r=\chi_0}\chi(f(a))\right) = \frac{1}{r}\sum_{\chi^r=\chi_0}\sum_{a\in\mathbb{F}_\varrho^*}\chi(f(a))$$
$$\leq \frac{\varrho}{r} + \frac{1}{r}\sum_{\substack{\chi^r=\chi_0\\ \chi\neq\chi_0}}\left|\sum_{a\in\mathbb{F}_\varrho}\chi(f(a))\right|.$$

Here we use the obvious fact that

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

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 rth power of another polynomial, and χ has order $r>1$, then

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

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

$$R_f^{(r)} \leq \frac{\varrho}{r} + \frac{r-1}{r}\cdot\frac{\varrho}{2} \leq \frac{3\varrho}{4},$$

since $r\geq 2$. □

6.2.2 Certifying specified powers over $\mathbb{F}_q[x]$

Theorem 6.1 allows us to detect when a polynomial $f\in\mathbb{F}_\varrho[x]$ is a perfect rth 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 rth power.

Then we have two cases to consider. If f is an rth power, then clearly $f(\alpha)$ is an rth power as well, and for any $\alpha\in\mathbb{F}_\varrho$, we always have $\xi=1$.

Otherwise, if f is not an rth power, Theorem 6.1 demonstrates that for at least $1/4$ of the elements of $\mathbb{F}_\varrho$, $f(\alpha)$ is not an rth power. Thus, for α chosen randomly from $\mathbb{F}_\varrho$ we would expect $\xi\neq 1$ with probability at least $1/4$.

This idea works whenever the size of the multiplicative group is a multiple of the power r. For coefficients in an arbitrary finite field $\mathbb{F}_q$, where $q-1$ is not divisible by r, we work in a suitably chosen extension finite extension field. First, the requirement that the characteristic of $\mathbb{F}_q$ is strictly greater than $\deg f$ means that $q=p^e$ for some prime p greater than $\deg f$.

Since r must be less than $\deg f$, this implies that $r \nmid p$, and therefore that $r \nmid q$. Then from Fermat's Little Theorem, we know that $r \mid (q^{r-1}-1)$ and so we construct an extension field $\mathbb{F}_{q^{r-1}}$ over $\mathbb{F}_q$ and proceed as above. Algorithm 6.1 gives a more formal presentation of this idea.

Algorithm 6.1: Perfect rth power over $\mathbb{F}_q$

Input: A prime power q, $f \in \mathbb{F}_q[x]$ of degree n such that $\text{char}(\mathbb{F}_q) < n \leq 1+\sqrt{q}/2$, $r \in \mathbb{N}$ a prime dividing n, and $\epsilon \in \mathbb{R}_{>0}$

Output: **true** if f is the rth power of a polynomial in $\mathbb{F}_q[x]$; otherwise **false** with probability at least $1-\epsilon$.

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** $\xi \neq 1$ **then return** ***false***
9 **return** ***true***

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, meaning always correct and probably fast. We modify this in the trivial way to a Monte Carlo algorithm that is always fast and probably correct. That is, we allow the algorithm to execute the specified number of operations, and if no answer has been returned after this time, the algorithm is halted and returns "fail". From the run-time analysis of the Las Vegas algorithm, the probability of failure is at most $1/2$, and the algorithm never returns an incorrect answer. This modification allows us to use Shoup's algorithm in our Monte Carlo algorithm. To obtain an irreducible Γ 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 $n \leq 1+\sqrt{q}/2$ (or equivalently that $q \geq 4(n-1)^2$) is not at all limiting. If this condition is not met, simply extend $\mathbb{F}_q$ with an extension of degree $v = \left\lceil \log_q(4(n-1)^2) \right\rceil$ and perform the algorithm over $\mathbb{F}_{q^v}$. At worst, each operation in $\mathbb{F}_{q^v}$ requires $O(\mathsf{M}(\log n))$ operations in $\mathbb{F}_q$, which will not be significant in the overall complexity.

Theorem 6.2. *Let q, f, n, r, ϵ be as in the input to the Algorithm 6.1. If f is a perfect rth power the algorithm always reports this. If f is not a perfect rth 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 ϵ. □

Theorem 6.3. *On inputs as specified, Algorithm 6.1 requires*

$$O((r\mathsf{M}(r)\log r\log q)\cdot\log(1/\epsilon))$$

operations in $\mathbb{F}_q$*, plus the cost to evaluate* $f(\alpha)$ *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 5–8 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 $O(\mathsf{M}(r))$ operations in $\mathbb{F}_q$, and we repeat the loop $O(\log(1/\epsilon))$ times. □

Combining these facts, we have that our Monte Carlo algorithm for perfect power testing is correct and works over any finite field of sufficiently large characteristic.

Corollary 6.4. *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*

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

operations in $\mathbb{F}_q$*, provided* $n>\mathrm{char}(\mathbb{F}_q)$*. 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* $1-\epsilon$.

The results here are counting field operations on an algebraic IMM. However, since the field is specified carefully as $\mathbb{F}_q$, and we know how to represent such elements on the integer side, we could demand all computation be performed on a normal IMM and count word operations. The consequences for the complexity would be that the cost as stated is increased by a factor of $O(\mathsf{N}(q))$, but every $\mathsf{M}(r)$ can be written simply as $O(r\log r)$, as discussed above.

6.2.3 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, at least with high probability. For any polynomial $g\in\mathsf{F}[x]$, let $\mathrm{disc}(g)=\mathrm{res}(g,g')$ be the discriminant of g (the resultant of g and its first derivative). It is well known that g is squarefree if and only if $\mathrm{disc}(g)\neq 0$ (see e.g., von zur Gathen and Gerhard, 2003, §15.2). Also define $\mathrm{lcoeff}(g)$ as the leading coefficient of g, the coefficient of the highest power of x in g. Finally, for $g\in\mathbb{Z}[x]$ and p a prime, denote by $g\operatorname{rem}p$ the unique polynomial in $\mathbb{F}_p[x]$ with all coefficients in g reduced modulo p.

Lemma 6.5. *Let* $f\in\mathbb{Z}[x]$ *and* $\tilde f=f/\gcd(f,f')$ *its squarefree part. Let* p *be a prime such that* $p\nmid\mathrm{disc}(\tilde f)$ *and* $p\nmid\mathrm{lcoeff}(f)$*. Then* f *is a perfect power in* $\mathbb{Z}[x]$ *if and only if* $f\operatorname{rem}p$ *is a perfect power in* $\mathbb{F}_p[x]$.

Proof. Clearly if f is a perfect power, then $f \operatorname{rem} p$ 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 $\deg(f_i \operatorname{rem} p) = \deg f_i$ for $1 \le i \le m$. Since $p \nmid \operatorname{disc}(\tilde{f})$, $\tilde{f} \operatorname{rem} p$ is squarefree (see (von zur Gathen and Gerhard, 2003), Lemma 14.1), and each of the $f_i \operatorname{rem} p$ must be pairwise relatively prime and squarefree for $1 \le i \le m$. Now suppose $f \operatorname{rem} p$ is a perfect rth power modulo p. Then we must have $r \mid s_i$ for $1 \le i \le m$. 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 ∞-norm of g as $\left\|g\right\|_\infty = \max_i |g_i|$. Similarly, we define the coefficient 1-norm of g as $\left\|g\right\|_1 = \sum_i |g_i|$, and 2-norm as $\left\|g\right\|_2 = \left(\sum_i |g_i|^2\right)^{1/2}$. With $f, \tilde{f}$ as in Lemma 6.5, $\tilde{f}$ divides f, so we can employ the factor bound of (Mignotte, 1974) to obtain

$$\left\|\tilde{f}\right\|_\infty \le 2^n \left\|f\right\|_2 \le 2^n \sqrt{n+1} \cdot \left\|f\right\|_\infty .$$

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

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

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

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

prime factors greater than $4(n-1)^2$ (we require the lower bound $4(n-1)^2$ to employ Theorem 6.1 without resorting to field extensions). Choose an integer $\gamma \ge 4(n-1)^2$ such that the number of primes between γ and 2γ is at least $4\mu + 1$. By (Rosser and Schoenfeld, 1962), Corollary 3, the number of primes in this range is at least $3\gamma/(5 \ln \gamma)$ for $\gamma \ge 21$.

Now let $\gamma \ge \max\{21\mu \ln \mu, 226\}$. It is easily confirmed that if $\mu \le 6$ and $\gamma \ge 226$, then $3\gamma/(5 \ln \gamma) > 4\mu + 1$. Otherwise, if $\mu \ge 7$, then $\ln(21 \ln \mu) < 2 \ln \mu$, so

$$\frac{\gamma}{\ln \gamma} \ge \frac{21\mu \ln \mu}{\ln \mu + \ln(21 \ln \mu)} > 7\mu,$$

and therefore $3\gamma/(5 \ln \gamma) > 21\mu/5 > 4\mu + 1$.

Thus, if $\gamma \ge \max\{21\mu \ln \mu, 226\}$, 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 $\log_2 p \in O(\log n + \log\log \left\|f\right\|_\infty)$ bits.

Theorem 6.6. *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 rth power, then Algorithm 6.2 always reports this. If f is not a perfect rth power, on any invocation of the algorithm, this is reported correctly with probability at least $1 - \epsilon$.*

Algorithm 6.2: Perfect rth power over $\mathbb{Z}$

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 rth power of a polynomial in $\mathbb{Z}[x]$; **false** otherwise

1 $\mu \leftarrow \left\lceil \left\lceil \log_2\left(2^{2n^2}(n+1)^{2n}\left\|f\right\|_\infty^{2n+1}\right)\right\rceil / \left\lfloor \log_2\left(4(n-1)^2\right)\right\rfloor \right\rceil$
2 $\gamma \leftarrow \max\{\lceil 21\mu\ln\mu\rceil, 4(n-1)^2, 226\}$
3 **for** *i from 1 to* $\lceil \log_2(1/\epsilon)\rceil$ **do**
4 $\quad p \leftarrow$ random prime in the range $\gamma \ldots 2\gamma$
5 $\quad$ **if** *Algorithm 6.1 returns* ***false*** *on input* $(p, f \operatorname{rem} p, r, 1/4)$ **then return** ***false***
6 **return** ***true***

Proof. If f is an rth power then so is $f \operatorname{rem} p$ for any prime p, and so is any $f(\alpha) \in \mathbb{F}_p$. Thus, the algorithm always reports that f is an rth power. Now suppose f is not an rth power. If $p \mid \operatorname{disc}(f)$ or $p \mid \operatorname{lcoeff}(f)$ it may happen that $f \operatorname{rem} p$ *is* an rth power. This happens with probability at most $1/4$ and we will assume that the worst happens in this case. When $p \nmid \operatorname{disc}(f)$ and $p \nmid \operatorname{lcoeff}(f)$, the probability that Algorithm 6.1 incorrectly reports that f is an rth power is also at most $1/4$, by our choice of parameter ϵ in the call on step 5. Thus, on any iteration of steps 3–5, the probability of finding that f is an rth power is at most $1/2$. The probability of this happening $\lceil \log_2(1/\epsilon)\rceil$ times is at most ϵ. □

Theorem 6.7. *On inputs as specified, Algorithm 6.2 requires*

$$O\Big(r^2 \log^2 r \cdot (\log n + \operatorname{loglog}\left\|f\right\|_\infty)^2 \cdot (\operatorname{loglog} n + \operatorname{logloglog}\left\|f\right\|_\infty) \cdot \log(1/\epsilon)\Big)$$

or $O\tilde{}(r^2 \cdot (\operatorname{size}(f))^2 \cdot \log(1/\epsilon))$ *word operations in the IMM model, plus the cost to evaluate* $f(\alpha) \bmod p$ *at* $O(\log(1/\epsilon))$ *points* $\alpha \in \mathbb{F}_p$ *for primes* p *with* $\log p \in O(\log n + \log\log\left\|f\right\|_\infty)$.

Proof. The number of operations required by each iteration is dominated by Step 5, for which $O(r\mathsf{M}(r)\log r \log p)$ operations in $\mathbb{F}_p$ are sufficient by Theorem 6.3. Since $\log p \in O(\log n + \log\log\left\|f\right\|_\infty)$, and using the multiplication algorithm of Section 1.4, we obtain the final complexity as stated. □

The cost of evaluating a t-sparse polynomial $f \in \mathbb{Z}[x]$ modulo a prime p is

$$O(t \cdot \operatorname{size}(\left\|f\right\|_\infty) \cdot \operatorname{size}(p) + t \log n \cdot \mathsf{N}(p))$$

word operations, which is $O\tilde{}(\operatorname{size}(f) \cdot \operatorname{size}(p))$. Furthermore, from the theorem, we see that each prime has size bounded by $\operatorname{size}(f)$. We then obtain the following corollary for t-sparse polynomials in $\mathbb{Z}[x]$.

Corollary 6.8. *Given* $f \in \mathbb{Z}[x]$ *of degree* n *and* $r \in \mathbb{N}$ *a prime dividing* n*, we can determine if* f *is an* r*th power with*

$$O\tilde{}\left(r^2 \cdot (\operatorname{size} f)^2 \cdot \log(1/\epsilon)\right)$$

machine word 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* $1-\epsilon$.

6.2.4 An upper bound on r

The algorithms we have seen so far require the power r to be known in advance. To adapt these to the general situation that r is not known, we show that r must be small compared to the sparse representation size of f, and therefore there are not too many "guesses" of r that we must make in order to solve the problem.

Specifically, we show in this subsection that if $f = h^r$ and $f \neq x^n$ then the value of r is polynomially bounded by $\tau(f)$. 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 size of coefficients in f.

Theorem 6.9. *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 $\|h\|_2 \leq \left\|f\right\|_1^{1/r}$.*

Proof. Let $p > n$ be prime and $\zeta \in \mathbb{C}$ a pth 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 $|f(\theta)| \leq \left\|f\right\|_1$, since $|\theta| = 1$. Thus,

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

The following corollary is particularly useful.

Corollary 6.10. *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) $r \leq 2 \log_2 \left\|f\right\|_1$,

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

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 $\left\|f\right\|_1 \leq \tau(f) \left\|f\right\|_\infty$.

Algorithm 6.3: Perfect power detection over $\mathbb{Z}$

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 P ← {primes r | n and r ≤ 2 log_2(t ||f||_∞)}
2 for r ∈ P do
3     if Algorithm 6.2 returns true on input (f, r, ε/#P) then return true and r
4 return false
```

6.2.5 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).

Theorem 6.11. *If $f \in \mathbb{Z}[x] = h^r$ for some $h \in \mathbb{Z}[x]$, then Algorithm 6.3 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 $\tilde{O}((\text{size}(f))^4 \cdot \log(1/\epsilon))$ word operations.*

Proof. From the preceding discussions, we can see that if f is a perfect power, then it must be a perfect rth power for some $r \in \mathscr{P}$. 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 rth power; the probability of this occurring is the probability of error when f is not a perfect power, and is less than $\epsilon/\#\mathscr{P}$ at each iteration. So the probability of error on any iteration is at most ϵ, which is what we wanted.

The complexity result follows from the fact that each $r \in \tilde{O}(\text{size}(f))$ and using Corollary 6.8. □

We now turn to the case of finite fields. Here we rely on Schinzel's bound that $r \leq t-1$, and obtain the following algorithm.

Algorithm 6.4: Perfect power detection over $\mathbb{F}_q$

Input: A prime power q, $f \in \mathbb{F}_q[x]$ of degree n and sparsity t such that $n < \text{char}(\mathbb{F}_q)$, 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 P ← {primes r | n and r ≤ t}
2 for p ∈ P do
3     if Algorithm 6.1 returns true on input (f, r, ε/#P) then
4         return true and r
5
6 return false
```

Theorem 6.12. *If $f = h^r$ for $h \in \mathbb{F}_q[x]$, then Algorithm 6.4 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 $\tilde{O}(t^3(\log q + \log n))$ operations in $\mathbb{F}_q$.*

Proof. The proof is equivalent to that of Theorem 6.11, using the complexity bounds in Corollary 6.4. □

6.2.6 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 (6.1), 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 6.13. *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*

$$\Delta = \mathrm{disc}_x(\tilde{f}(y_1x,\ldots,y_\ell x)) = \mathrm{res}_x(\tilde{f}(y_1x,\ldots,y_\ell x), \tilde{f}'(y_1x,\ldots,y_\ell x)) \in \mathsf{F}[y_1,\ldots,y_\ell]$$

and

$$\Lambda = \mathrm{lcoeff}_x(f(y_1x,\ldots,y_\ell x)) \in \mathsf{F}[y_1,\ldots,y_\ell]$$

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 $f(a_1x,\ldots,a_\ell x) \in \mathsf{F}[x]$ is a perfect power.

Proof. Clearly if f is a perfect power, then $f(a_1x,\ldots,a_\ell x)$ 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

$$f(y_1x,\ldots,y_mx) = f_1(y_1x,\ldots,y_mx)^{s_1} \cdots f_m(y_1x,\ldots,y_mx)^{s_m}$$

and each of the $f_i(y_1x,\ldots,y_mx)$ are irreducible. Now, since $\Lambda(a_1,\ldots,a_m) \neq 0$, we know the $\deg(f(a_1x,\ldots,a_\ell x)) = \deg f$ (the total degree of f). Thus, $\deg f_i(a_1x,\ldots,a_\ell x) = \deg f_i$ for $1 \leq i \leq \ell$ as well. Also, by our assumption, $\mathrm{disc}(f(a_1x,\ldots,a_\ell x)) \neq 0$, so all of the $f_i(a_1x,\ldots,a_\ell x)$ are squarefree and pairwise relatively prime for $1 \leq i \leq k$, and

$$f(a_1x,\ldots,a_\ell x) = f_1(a_1x,\ldots,a_\ell x)^{s_1} \cdots f_m(a_1x,\ldots,a_\ell x)^{s_m}.$$

Assume now that $f(a_1x,\ldots,a_\ell x)$ is an rth perfect power. Then r divides s_i for $1 \leq i \leq m$. This immediately implies that f itself is an rth perfect power. □

It is easy to see that the total degree of Δ is less than $2n^2$ and the total degree of Λ is less than n, and that both Δ and Λ are non-zero. Thus, for randomly chosen $a_1,\ldots,a_\ell$ from a set $\mathscr{S} \subseteq \mathsf{F}$ 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 the famous Schwartz-Zippel lemma (Demillo and Lipton, 1978; Zippel, 1979;

Schwartz, 1980). This can be made arbitrarily small by increasing the set size and/or by repetition. We then run the appropriate univariate algorithm over $\mathsf{R}[x]$ (depending upon the ring) to identify whether or not f is a perfect power, and if so, to find r.

For integer polynomials, $f(a_1x,\ldots,a_\ell x)$ will *not* be explicitly computed over $\mathbb{Z}[x]$, as the size of the coefficients in this univariate polynomial could be exponentially larger than the sparse representation size of f itself. Instead, we pass $f(a_1x,\ldots,a_\ell x)$ unevaluated to Algorithm 6.2, which only performs the computation once a small finite field has been chosen. This preserves polynomial-time for sparse multivariate integer polynomials.

6.3 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) shows 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 F is not too small, $\tau(h) \in O(\tau(h^r)+r)$. The first algorithm presented here avoids this problem by being output-sensitive. That is, the cost of the algorithm is proportional to $\tau(h)$, whatever that might be. We then present different algorithm for this problem that will be much more efficient in practice, but whose analysis relies on a modest conjecture.

6.3.1 Computing rth roots in output-sensitive polynomial time

In this subsection we present an algorithm for computing an h such that $f = h^r$ given $f \in \mathbb{Z}[x]$ and $r \in \mathbb{Z}$ or showing that no such h exists. The algorithm is deterministic and requires time polynomial in size(f) as well as a given upper bound μ on $m = \tau(h)$. Neither its correctness nor complexity is conditional on any conjectures. We will only demonstrate that this algorithm requires polynomial time, as the (conjecturally fast) algorithm of the next chapter will be the obvious choice in practical situations.

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 cyclotomic field over $\mathbb{Q}$. Doing this for a relatively small number of cyclotomic fields yields h.

Assume that (the unknown) $h \in \mathbb{Z}[x]$ is written as

$$h = b_1x^{d_1} + b_2x^{d_2} + \cdots + b_mx^{d_m},$$

for $b_1,\dots,b_m \in \mathbb{Z}^*$, and $0 \le d_1 < d_2 < \cdots < d_m$,. Also assume that $p > 2$ is a prime distinct from r such that

$$p \nmid \prod_{1\le i<j\le m} (d_j - d_i), \text{ and } p \nmid \prod_{1\le i\le m} (d_i + 1). \tag{6.5}$$

Let $\zeta_p \in \mathbb{C}$ be a pth primitive root of unity, and $\Phi_p = 1 + z + \cdots + z^{p-1} \in \mathbb{Z}[z]$ its minimal polynomial, the pth cyclotomic polynomial (which is irreducible in $\mathbb{Q}[z]$). Computationally we represent $\mathbb{Q}(\zeta_p)$ as $\mathbb{Q}[z]/(\Phi_p)$, with $\zeta_p \equiv z \operatorname{rem} \Phi_p$. 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, if we define

$$h_p = \sum_{1\le i\le m} b_i x^{d_i \operatorname{rem} p} \in \mathbb{Z}[x],$$

then $h(\zeta_p) = h_p(\zeta_p)$, and h_p is the unique representation of $h(\zeta_p)$ as a polynomial of degree less than $p-1$. This follows from the conditions (6.5) on our choice of prime p because

- No pair of distinct exponents d_i and d_j of h is equivalent modulo p (since $p \nmid (d_i - d_j)$), 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 $1 \le i \le m$).

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 the algebraic extension field $\mathbb{Q}(\zeta_p)[y]$, and the roots in $\mathbb{C}$ must be $\omega^i h(\zeta_p) \in \mathbb{C}$ for $0 \le i < r$, where ω is a primitive rth root of unity. When $r > 2 h_p = \sum_{1\le i\le m} b_i x^{d_i \operatorname{rem} p} \in \mathbb{Z}[x]$,, and since we chose p distinct from r, the only rth root of unity in $\mathbb{Q}(\zeta_p)$ is 1. Thus $\Gamma_p(y)$ has exactly one linear factor, and this must be 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 ± 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 repeatedly perform the above operations for a sequence of cyclotomic fields $\mathbb{Q}(\zeta_{p_1}), \mathbb{Q}(\zeta_{p_2}), \dots, \mathbb{Q}(\zeta_{p_k})$ such that the primes in $\mathscr{P} = \{p_1,\dots,p_k\}$ allow us to recover all the exponents in h. Each prime $p \in \mathscr{P}$ 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 $p \in \mathscr{P}$ we obtain

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

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, 2009) (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 $p \in \mathscr{P}$ we compute the symmetric polynomial modulo p,

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

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 $p \in \mathscr{P}$ must be at least $2\left\|g\right\|_\infty$ to recover the coefficients of g uniquely. It is easily seen that $2\left\|g\right\|_\infty \le 2n^m$.

As noted above, the computation with each $p \in \mathscr{P}$ recovers all the exponents of h in $\mathbb{Z}$, so using only one prime $p \in \mathscr{P}$, we determine the jth exponent of h as the coefficient of $x^{d_j \operatorname{rem} p}$ in h_p for $1 \le j \le m$. 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.

Finally, once we have a candidate root h, we certify that $f = h^r$ by taking logarithmic derivatives to obtain

$$\frac{f'}{f} = \frac{r h' h^{r-1}}{h^r},$$

which simplifies to $f'h = rh'f$. This equation only involves two sparse multiplications and is therefore confirmed in polynomial time, and along with checking leading coefficients implies that in fact $f = h^r$.

The above discussion is summarized in the following algorithm.

Theorem 6.14. *Algorithm 6.5 works as stated. It requires a number of word operations polynomial in* size(f) *and* μ.

Proof. We assume throughout the proof that there *does* exist an $h \in \mathbb{Z}[x]$ such that $f = h^r$ and $\tau(h) \le \mu$. If it does not, this will be caught in the test in Steps 22–24 by the above discussion, if not before.

In Steps 1–2 we construct a set of primes $\mathscr{P}$ which is guaranteed to contain sufficiently many *good* primes to recover g, where primes are good in the sense that for all $p \in \mathscr{P}$

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

It is easily derived that $\beta < n^{\mu^2}$, which has fewer than $\log_2 \beta \le \mu^2 \log_2 n$ prime factors, so there are at most $\mu^2 \log_2 n$ *bad* primes. We also need to recover g in Step 16, and $\left\|g\right\|_\infty \le n^\mu$, for which we need at least $1 + \log_2 \left\|g\right\| \le 2\mu \log_2 n$ good primes. Thus if $\mathscr{P}$ has at least $(\mu^2 + 2\mu)\log_2 n$ primes, there are a sufficient number of good primes to reconstruct g in Step 16.

Algorithm 6.5: Algebraic algorithm for computing perfect roots

Input: $f \in \mathbb{Z}[x]$ as in (6.2) with $\deg f = n$, 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

$\gamma \leftarrow$ smallest integer ≥ 21 such that $3\gamma/(5\ln\gamma) \ge (\mu^2+2\mu)\log_2 n$
$\mathscr{P} \leftarrow \{p \in \{\gamma, \ldots, 2\gamma\}$ and p prime$\}$
for $p \in \mathscr{P}$ **do**
    Represent $\mathbb{Q}(\zeta_p)$ by $\mathbb{Q}[x]/(\Phi_p)$, where $\Phi_p \leftarrow 1 + z + \cdots + z^{p-1}$ and $\zeta_p \equiv z \bmod \Phi_p$
    Compute $f(\zeta_p) = \sum_{1 \le i \le t} c_i \zeta_p^{e_i \operatorname{rem} p} \in \mathbb{Z}[\zeta_p]$
    Factor $\Gamma_p(y) \leftarrow y^r - f(\zeta_p) \in \mathbb{Q}(\zeta_p)[y]$ over $\mathbb{Q}(\zeta_p)[y]$
    if $\Gamma_p(y)$ *has no roots in* $\mathbb{Z}[\zeta_p]$ **then**
        return *"f is not an rth power of a μ-sparse polynomial"*
    Let $h_p(\zeta_p) \in \mathbb{Z}[\zeta_p]$ be a root of $\Gamma_p(y)$
    Write $h_p(x) = \sum_{1 \le i \le m_p} b_{ip} x^{d_{ip}}$, for $b_{ip} \in \mathbb{Z}$ and distinct $d_{ip} \in \mathbb{N}$ for $1 \le i \le m_p$
    if $\deg h_p = p - 1$ **then**
        $m_p \leftarrow 0$; Continue with next prime $p \in \mathscr{P}$ at Step 3
    $g_p \leftarrow (x - d_{1p})(x - d_{2p}) \cdots (x - d_{m_p p}) \in \mathbb{Z}_p[x]$
$m \leftarrow \max\{m_p : p \in \mathscr{P}\}$
$\mathscr{P}_0 \leftarrow \{p \in \mathscr{P} : m_p = m\}$
Reconstruct $g \in \mathbb{Z}[x]$ from $\{g_p\}_{p \in \mathscr{P}_0}$ by the Chinese Remainder Algorithm
$\{d_1, d_2, \ldots, d_k\} \leftarrow$ distinct integer roots of g
if $k < m$ **then**
    return *"f is not an rth power of a μ-sparse polynomial"*
Choose any $p \in \mathscr{P}_0$. For $1 \le j \le m$, let $b_j \in \mathbb{Z}$ be the coefficient of $x^{d_j \operatorname{rem} p}$ in h_p
$h \leftarrow \sum_{1 \le j \le m} b_j x^{d_j}$
if $f'h = rh'f$ ***and*** $\operatorname{lcoeff}(f) = \operatorname{lcoeff}(h)^r$ **then**
    return h
else
    return *"f is not an rth power of a μ-sparse polynomial"*

By (Rosser and Schoenfeld, 1962), Corollary 3, for $\gamma \geq 21$ we have that the number of primes in $\{\gamma, \ldots, 2\gamma\}$ is at least $3\gamma/(5\ln\gamma)$, which is at least $(\mu^2+2\mu)\log_2 n$ by our choice of γ in Step 1, and $\gamma \in O(\mu^2 \log(n))$. Numbers of this size can easily be tested for primality.

Since we assume that a root h exists, $\Gamma_p(y)$ will always have exactly one root $h_p \in \mathbb{Z}[\zeta_p]$ when $r > 2$, and exactly two roots in $\mathbb{Z}[\zeta_p]$ when $r = 2$ (differing only by sign).

Two conditions cause the primes to be identified as bad. If the map $h \mapsto h(\zeta_p)$ causes some exponents of h to collide modulo p, this can only reduce the number of non-zero exponents m_p in h_p, and so such primes will not show up in the list of good primes $\mathscr{P}_0$, as selected in Step 15. Also, if any of the exponents of h are equivalent to $p-1$ modulo p we will not be able to reconstruct the exponents of h from h_p, and we identify these as bad in Step 12 (by artificially marking $m_p = 0$, which ensures they will not be added to $\mathscr{P}_0$).

Correctness of the remainder of the algorithm follows from the previous discussion.

The complexity is clearly polynomial for all steps except for factoring in $\mathbb{Q}(\zeta_p)[y]$ (Step 6). It is well-established that factoring dense polynomials over algebraic extensions can be done in polynomial time, for example using the algorithm of Landau (1985). □

As stated, Algorithm 6.5 is not actually output-sensitive, as it requires an *a priori* bound μ on $\tau(h)$. To avoid this, we could start with any small value for μ, say $\tau(f)$, and after each failure double this bound. Provided that the input polynomial f is in fact an rth perfect power, this process with terminate after a number of steps polynomial in the sparse size of the output polynomial h. There are also a number of other small improvements that could be made to increase the algorithm's efficiency, which we have omitted here for clarity.

6.3.2 Faster root computation subject to conjecture

Algorithm 6.5 is output sensitive as the cost depends on the sparsity of the root h. As discussed above, there is considerable evidence that, roughly speaking, the root of a sparse polynomial must always be sparse, and so the preceding algorithm may be unconditionally polynomial-time.

In fact, with suitable sparsity bounds we can derive a more efficient algorithm based on Newton iteration. This approach is simpler as it does not rely on advanced techniques such as factoring over algebraic extension fields. 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 a conjecture regarding the sparsity of powers of h. We first present the algorithm and prove its correctness. Then we give our modest conjecture and use it to prove the algorithm's efficiency.

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 $i = 1, 2, \ldots, [\log_2(\deg h + 1)]$, 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 F is either zero or greater than n. As usual, we define $t = \tau(f)$.

Algorithm 6.6: Sparsity-sensitive Newton iteration to compute perfect roots

Input: $f \in \mathsf{F}[x]$, $r \in \mathbb{N}$ such that f is a perfect rth 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 $\quad a \leftarrow \dfrac{(hg - h^{r+1}) \operatorname{rem} x^{k+\ell}}{r x^k}$
8 $\quad h \leftarrow h + (a/g \bmod x^\ell) \cdot x^k$
9 $\quad k \leftarrow k + \ell$
10 $b \leftarrow$ any rth root of f_u in F
11 **return** $bhx^{u/r}$

Theorem 6.15. *If $f \in \mathsf{F}[x]$ is a perfect rth power, then 6.6 returns an $h \in \mathsf{F}[x]$ such that $h^r = f$.*

Proof. Let u, f_u, g be as defined in Steps 1–3. Thus $f = f_u g x^u$. Now let $\hat{h}$ be some rth 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 rth power in F, $r|u$, and g is a perfect rth 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 ith 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 jth iteration of Step 7. We observe that

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

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

$$h_j h_{j+1}^r \equiv h_j^{r+1} + r a x^k \equiv h_j^{r+1} + h_j f - h_j^{r+1} \equiv h_j f \pmod{x^{k+\ell}}.$$

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 rth root of g. Finally, $\left(bhx^{u/r}\right)^r = f_u g x^u = f$. □

Algorithm 6.6 will only be efficient if the low-order terms of the polynomial power h^{r-1} can be efficiently computed on Step 7. Since we know that h and the low-order terms of h^{r-1} are sparse, we need only a guarantee that the *intermediate powers* will be sparse as well. This is stated in the following modest conjecture.

Conjecture 6.16. *For $r, s \in \mathbb{N}$, if the characteristic of 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 constrained with only s degrees of freedom. Computationally, the conjecture has also been confirmed for all of the numerous examples we have tested, and for the special case of $r = 2$ and $s \leq 12$ by Abbott (2002), although a more thorough investigation of its truth would be interesting. A weaker inequality would suffice to prove polynomial time, but we use the stated bounds as we believe these give more accurate complexity measures.

The application of Conjecture 6.16 to Algorithm 6.6 is given by the following lemma, which essentially tells us that the "error" introduced by examining higher-order terms of h_1^r is not too dense.

Lemma 6.17.[2] *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 ℓ 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 6.16 is less than $t - r$, the total number of terms in $h_1^{r-1} \bmod x^{k+\ell}$ is less than $2t(t-r)$. □

We are now ready to prove the efficiency of the algorithm, assuming the conjecture.

Theorem 6.18.[2] *If $f \in \mathsf{F}[x]$ has degree n and t nonzero terms, then Algorithm 6.6 uses*

$$O\left((t+r)^4 \log r \log n\right)$$

operations in F and an additional $O\left((t+r)^4 \cdot \text{size}(r) \cdot \log^2 n\right)$ word operations, not counting the cost of root-finding in the base field F on Step 10.

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 6.16. The number of field operations required, at each iteration, is $O\left((t+r)^4 \log r\right)$, for a total cost of $O\left((t+r)^4 \log r \log n\right)$.

[2] Subject to the validity of Conjecture 6.16.

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

$$\sum_{1\le i<\log_2 n} (t+r)^4 \cdot \text{size}(r\,i) \in O\left((t+r)^4 \,\text{size}(r)\log^2 n\right).$$

In fact, this is the most costly step. The initialization in Steps 1–3 uses only $O(t)$ operations in 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. Observe that the difference in the number of word operations on an IMM and the number of bit operations for the same algorithm is at most a factor of the word size w. From the definition of an IMM, we know that this is logarithmic in the size of the input, and therefore does not affect the $\tilde{O}(\cdots)$ complexity.

So for ease of presentation, and in particular in order to make use of Theorem 6.9, we will count the cost of the computation in bit complexity. First we define the height of a polynomial in terms of bit length: For $\alpha \in \mathbb{Q}$, write $\alpha = a/b$ for a, b relatively prime integers. Then define $\mathscr{H}(\alpha) = \max\{|a|,|b|\}$. And for $f \in \mathbb{Q}[x]$ with coefficients $c_1,\ldots,c_t \in \mathbb{Q}$, write $\mathscr{H}(f) = \max \mathscr{H}(c_i)$.

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

Theorem 6.19.[2] *Suppose $f \in \mathbb{Q}[x]$ has degree n and t nonzero terms, and is a perfect r th power. Algorithm 6.6 computes an r th root of f using $\tilde{O}\left(t(t+r)^4 \cdot \log n \cdot \log \mathscr{H}(f)\right)$ 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 6.9, we have:

$$\mathscr{H}(h) \le \|ch\|_\infty \le \|ch\|_2 \le (t\left\|c^r f\right\|_\infty)^{1/r} \le t^{1/r}\mathscr{H}(f)^{(t+1)/r}.$$

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

Hence $\log \mathscr{H}(h) \in O\left((t\log \mathscr{H}(f))/r\right)$. 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 ith iteration of the while loop in our algorithm) as equal to h (the *actual* root of f), since we know $\tau(h_i) \le \tau(h)$ and $\mathscr{H}(h_i) \le \mathscr{H}(h)$.

Lemma 6.17 and Conjecture 6.16 tell us that $\tau(h^i) \le 2(t+r)^2$ 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.

[2] Subject to the validity of Conjecture 6.16.

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

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

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

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

Again, observe that the bit complexity in general is an upper bound on the number of IMM operations, and when we use the blunt measure of $O\tilde{}(\cdots)$ notation, the costs are exactly the same.

The method used for Step 10 depends on the field 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 $x^r - f_u$.

6.3.3 Computing multivariate roots

For the problem of computing perfect polynomial roots of multivariate polynomials, we again reduce the problem to a univariate one, this time employing the well-known Kronecker substitution method.

Suppose $f, h \in \mathsf{F}[x_1,\ldots,x_\ell]$ and $r \in \mathbb{N}$ such that $f = h^r$. It is easily seen that each partial degree of f is exactly r times the corresponding partial degree in h, that is, $\deg_{x_i} f = r \deg_{x_i} h$, for all $r \in \{1,\ldots,\ell\}$.

Now suppose f and r are given and we wish to compute h. First use the relations above to compute $d_i = \deg_{x_i} h + 1$ for each $i \in \{1,\ldots,\ell\}$. (If any $\deg_{x_i} f_i$ is not a multiple of r, then f must not be an rth power.)

Now use the Kronecker substitution and define

$$\hat{f} = f\left(y, y^{d_1}, y^{d_1 d_2}, \ldots, y^{d_1\cdots d_{\ell-1}}\right) \quad \text{and} \quad \hat{h} = h\left(y, y^{d_1}, y^{d_1 d_2}, \ldots, y^{d_1\cdots d_{\ell-1}}\right),$$

where y is a new variable. Clearly $\hat{f} = \hat{h}^r$, and since each $d_i > \deg_{x_i} h$, h is easily recovered from the sparse representation of $\hat{h}$ in the standard way: For each non-zero term $c\,y^e$ in $\hat{h}$, compute the digits of e in the mixed radix representation corresponding to the sequence $d_1, d_2, \ldots, d_{\ell-1}$. That is, decompose e (uniquely) as $e = e_1 + e_2 d_1 + e_3 d_1 d_2 + \cdots + e_\ell d_1 \cdots d_{\ell-1}$ with each $e_i \in \mathbb{N}$ such that $e_i < d_i$. Then the corresponding term in h is $c\,x_1^{e_1}\cdots x_\ell^{e_\ell}$.

Therefore we simply use either algorithm above to compute $\hat{h}$ as the rth root of $\hat{f}$ over $\mathsf{F}[y]$, then invert the Kronecker map to obtain $h \in \mathsf{F}[x_1,\ldots,x_\ell]$. The conversion steps are clearly polynomial-time, and notice that $\log\deg\hat{f}$ is at most ℓ times larger than $\log\deg f$. Therefore the lacunary sizes of $\hat{f}$ and $\hat{h}$ are polynomial in the lacunary sizes of f and h, and the algorithms in this section yield polynomial-time algorithms to compute perfect rth roots of multivariate lacunary polynomials.

6.4 Implementation

To investigate the practicality of our algorithms, we implemented Algorithm 6.3 using NTL (Shoup, 2009), for dense univariate polynomials with arbitrary-precision integers as their coefficients. The only significant difference between our implementation and the algorithm specified above 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 $(r-1) \mid p$. Proving good complexity bounds is more difficult in this situation, as it requires bounds on primes in arithmetic progressions. That issue in particular will be discussed much more thoroughly in Chapter 8, but for now we simply point out that this is extremely efficient in practice, as it avoids working in extension fields.

To our knowledge, all existing algorithms which could be used to test for perfect powers actually compute the root h. We implemented the fastest method for densely-represented polynomials, both in theory and in practice, which is a standard Newton iteration. This algorithm uses dense polynomial arithmetic in its implementation, and also requires the power r to be given explicitly. This Newton iteration method is adapted to the case when r is not known by simply trying all possible powers r, and then checking with a single evaluation whether the computed candidate h is actually an rth root of f. This technique is not provably correct, but in all of our trials it has never failed. Furthermore, since this is the algorithm we are comparing *against*, we graciously give it every possible advantage.

The results of these experiments were reported in (Giesbrecht and Roche, 2008). We found that, when the given polynomial f was *not* a perfect power, our algorithm detected this extremely quickly, in fact always with just one random evaluation. Even when the inputs had mostly nonzero coefficients, we found that our algorithm performed competitively with the dense Newton iteration approach. This is due to the black-box nature of the algorithm — it requires only a way to quickly evaluate a polynomial at a given point, which is of course possible whether the polynomial is sparsely or densely represented. Much more on the black box representation and what other kinds of information can be learned from it will be discussed in the next two chapters.

6.5 Conclusions

Given a sparse polynomial over the integers, rational numbers, or a finite field, we have shown polynomial-time Monte Carlo algorithms to determine whether the polynomial is a perfect rth power for any $r \geq 2$. In every case, the error is one-sided, i.e., the algorithm may produce false positives but never true negatives. Therefore we have shown that these problems are in the complexity class **coRP**.

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. We presented an output-sensitive algorithm that makes use of factorization over algebraic extension fields, and also a conjecturally-fast sparse Newton iteration.

There are many interesting potential connections from our algorithms to other related problems. First, polynomial perfect powers are a special case of the functional decomposition problem. Given a polynomial $f \in \mathsf{R}[x]$, the univariate version of this problem asks for a pair of polynomials $g, h \in \mathsf{R}[x]$ such that $f(x) = g(h(x))$. This connection is made more explicit in our discussion of Algorithm 6.6 in the next chapter. For now, it suffices to say that it is not known whether decomposition can be solved in polynomial time for sparse polynomials.

Perfect powers are also a special case of factorization. As mentioned in the beginning of this chapter, existing algorithms for factorization of sparse polynomials, for example those by Lenstra (1999); Kaltofen and Koiran (2006), require polynomial-time in the degree of the computed factors. Because the degree of a sparse polynomial can be exponential in the sparse representation size, this precludes the computation of high-degree factors which have few nonzero terms. The combination of Algorithms 6.3 and 6.5 gives the first polynomial-time algorithm to compute *any* sparse, high-degree factors of a sparse polynomial. A rich topic of further investigation would be the development of further algorithms of this type, with the ultimate goal being an either an algorithm to compute all t-sparse factors in $t^{O(1)}$ time, or a reduction showing the general problem to be intractable.

Bibliography

John Abbott. Sparse squares of polynomials. *Math. Comp.*, 71(237):407–413, 2002. ISSN 0025-5718.
doi: 10.1090/S0025-5718-00-01294-1. Referenced on pages 106 and 112.

Eric Bach and Jonathan Sorenson. Sieve algorithms for perfect power testing. *Algorithmica*, 9:313–328, 1993. ISSN 0178-4617.
doi: 10.1007/BF01228507. Referenced on page 95.

Daniel J. Bernstein. Detecting perfect powers in essentially linear time. *Math. Comp.*, 67(223): 1253–1283, 1998. ISSN 0025-5718.
doi: 10.1090/S0025-5718-98-00952-1. Referenced on pages 95 and 114.

Don Coppersmith and James Davenport. Polynomials whose powers are sparse. *Acta Arith*, 58:79–87, 1991. Referenced on page 106.

Felipe Cucker, Pascal Koiran, and Steve Smale. A polynomial time algorithm for Diophantine equations in one variable. *J. Symbolic Comput.*, 27(1):21–29, 1999. ISSN 0747-7171.
doi: 10.1006/jsco.1998.0242. Referenced on page 95.

Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978. ISSN 0020-0190.
doi: 10.1016/0020-0190(78)90067-4. Referenced on page 105.

P. Erdös. On the number of terms of the square of a polynomial. *Nieuw Arch. Wiskunde (2)*, 23:63–65, 1949. Referenced on page 106.

Sanchit Garg and Éric Schost. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science*, 410(27-29):2659–2662, 2009. ISSN 0304-3975.
doi: 10.1016/j.tcs.2009.03.030. Referenced on page 108.

Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, second edition, 2003. ISBN 0521826462. Referenced on pages 100 and 101.

Joachim von zur Gathen, Marek Karpinski, and Igor Shparlinski. Counting curves and their projections. *Computational Complexity*, 6:64–99, 1996. ISSN 1016-3328.
doi: 10.1007/BF01202042. Referenced on page 94.

Mark Giesbrecht and Daniel S. Roche. On lacunary polynomial perfect powers. In *ISSAC '08: Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 103–110, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-904-3. doi: 10.1145/1390768.1390785. Referenced on pages 94 and 115.

Mark Giesbrecht and Daniel S. Roche. Detecting lacunary perfect powers and computing their roots. *Journal of Symbolic Computation*, to appear, 2011. URL http://arxiv.org/abs/0901.1848. Referenced on page 94.

Dima Grigoriev, Marek Karpinski, and Andrew M. Odlyzko. Short proofs for nondivisibility of sparse polynomials under the extended Riemann hypothesis. *Fundam. Inf.*, 28(3-4):297–301, 1996. ISSN 0169-2968. Referenced on page 94.

Dima Yu. Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 166–172, October 1987. doi: 10.1109/SFCS.1987.56. Referenced on page 108.

E. Kaltofen. Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 443–452, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: 10.1145/28395.28443. Referenced on page 95.

Erich Kaltofen and Pascal Koiran. On the complexity of factoring bivariate supersparse (lacunary) polynomials. In *ISSAC '05: Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 208–215, New York, NY, USA, 2005. ACM. ISBN 1-59593-095-7. doi: 10.1145/1073884.1073914. Referenced on page 95.

Erich Kaltofen and Pascal 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, New York, NY, USA, 2006. ACM. ISBN 1-59593-276-3. doi: 10.1145/1145768.1145798. Referenced on pages 95 and 116.

Marek Karpinski and Igor Shparlinski. On the computational hardness of testing squarefreeness of sparse polynomials. In Marc Fossorier, Hideki Imai, Shu Lin, and Alain Poli, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 1719 of *Lecture Notes in Computer Science*, pages 731–731. Springer Berlin / Heidelberg, 1999. doi: 10.1007/3-540-46796-3_47. 10.1007/3-540-46796-3_47. Referenced on page 95.

Susan Landau. Factoring polynomials over algebraic number fields. *SIAM Journal on Computing*, 14(1):184–195, 1985. doi: 10.1137/0214015. Referenced on page 110.

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. Referenced on pages 95 and 116.

Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Company Advanced Book Program, Reading, MA, 1983. ISBN 0-201-13519-1. Referenced on pages 97 and 98.

M. Mignotte. An inequality about factors of polynomials. *Math. Comp.*, 28:1153–1157, 1974. ISSN 0025-5718. Referenced on page 101.

David A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theoret. Comput. Sci.*, 31(1-2):125–138, 1984. ISSN 0304-3975. doi: 10.1016/0304-3975(84)90130-0. Referenced on page 94.

David Alan Plaisted. Sparse complex polynomials and polynomial reducibility. *J. Comput. System Sci.*, 14(2):210–221, 1977. ISSN 0022-0000. Referenced on page 94.

Andrew Quick. Some GCD and divisibility problems for sparse polynomials. Master's thesis, University of Toronto, 1986. Referenced on page 94.

J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962. URL http://projecteuclid.org/euclid.ijm/1255631807. Referenced on pages 101 and 110.

A. Schinzel. On the number of terms of a power of a polynomial. *Acta Arith.*, 49(1):55–70, 1987. ISSN 0065-1036. Referenced on pages 103 and 106.

J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27:701–717, October 1980. ISSN 0004-5411. doi: 10.1145/322217.322225. Referenced on page 106.

Victor Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994. ISSN 0747-7171. doi: 10.1006/jsco.1994.1025. Referenced on pages 99 and 100.

Victor Shoup. NTL: A Library for doing Number Theory. Online, August 2009. URL http://www.shop.net/ntl/. Version 5.5.2. Referenced on page 115.

Igor E. Shparlinski. Computing Jacobi symbols modulo sparse integers and polynomials and some applications. *Journal of Algorithms*, 36(2):241–252, 2000. ISSN 0196-6774. doi: 10.1006/jagm.2000.1091. Referenced on page 95.

André Weil. On some exponential sums. *Proc. Nat. Acad. Sci. U. S. A.*, 34:204–207, 1948. ISSN 0027-8424. Referenced on page 98.

David Y. Y. Yun. On square-free decomposition algorithms. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, SYMSAC '76, pages 26–35, New York, NY, USA, 1976. ACM. doi: 10.1145/800205.806320. Referenced on page 95.

Umberto Zannier. On the number of terms of a composite polynomial. *Acta Arith,* 127(2): 157–167, 2007.
doi: 10.4064/aa127-2-5. Referenced on page 106.

Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward Ng, editor, *Symbolic and Algebraic Computation,* volume 72 of *Lecture Notes in Computer Science,* pages 216–226. Springer Berlin / Heidelberg, 1979.
doi: 10.1007/3-540-09519-5_73. Referenced on page 105.