The Sparsity Challenges

(Invited Paper)

James H. Davenport
Department of Computer Science
University of Bath
Bath BA2 7AY, United Kingdom
Email: J.H.Davenport@bath.ac.uk

Jacques Carette
Department of Computing and Software
McMaster University
Hamilton, Ontario
Email: carette@mcmaster.ca

***Abstract*—While much is written about the importance of sparse polynomials in computer algebra, much less is known about the complexity of advanced (i.e. anything more than multiplication!) algorithms for them. This is due to a variety of factors, not least the problems posed by cyclotomic polynomials. In this paper we state a few of the challenges that sparse polynomials pose.**

I. Introduction

Most computer algebra texts (e.g. [1], [2]) state that any realistic representation of polynomials has to be sparse, and then carefully explain how to add and multiply sparse polynomials, often citing [3], with complexity measured in terms of the number of monomials. They then discuss algorithms for division, g.c.d., factorization etc., while implicitly adopting the dense paradigm when it comes to discussing their running time etc., even though the underlying implementations may well still be sparse. The first author confesses to this lapse, which was, at the time, unconscious. Sometimes black box representations are mentioned, but they too quickly fall out of sight.

So why do authors do this? The easy answer is that there are no good ways of doing these operations with sparse polynomials, but that is too facile. The purpose of this paper is to delve somewhat deeper into this challenge: that of producing

> algorithms which manipulate sparse polynomials in ways that respect their sparsity, and whose complexity is "reasonable" in terms of the (sparse) size of the inputs.

In fact it turns out that each operation we may wish to perform has its own difficulties here, hence the title refers to "challenges" in the plural. We largely restrict ourselves to univariate polynomials, since if the problems cannot be solved for that case, there is apparently little hope elsewhere.

A. Notation

We assume our polynomial variable is x, and a polynomial f has degree d_f and t_f non-zero terms, *mutatis mutandis* for other polynomials. So $t_f \leq d_f + 1$, but we are typically concerned with cases where t_f is much smaller than d_f, and we will refer to the ratio (strictly speaking $t_f/(d_f+1)$ as the "sparsity" of f. Let $|f|$ be the largest coefficient in absolute value in f.

Our coefficients will always lie in an integral domain, and unless otherwise stated, one of characteristic zero. Our "standard model" is the typical "sparse polynomial" representation of a list of (exponent, coefficient) pairs, as in [1, pp. 81–83]. It should be noted that we are only considering *explicit* integer exponents, though there is a lot to be said for learning from the case of symbolic exponents as well [4]–[6]. Other representations, such as sparse Horner form [7] and "Black Box" models [8], are also well worth investigating

We let Φ_n be the n-th cyclotomic polynomial, i.e. the polynomial whose roots are all the n-th roots of unity which are not lesser roots:

$$\Phi_n(x) = \prod_{\substack{k=1 \\ \gcd(k,n)=1}}^{n} \left(x - e^{2\pi i k/n}\right). \tag{1}$$

We let $\phi(n) = d_{\Phi_n}$, noting that this is the usual Eulerian ϕ function. If we let C_n be the polynomial $x^n - 1$, we see that these are related by the following result.

Proposition 1: $C_n(x) = \prod_{d|n} \Phi_d(x)$ and $\Phi_n(x) = \prod_{d|n} C_d(x)^{\mu(n/d)}$, where μ is the Möbius function.

The coefficients of Φ_n, which naïvely seem to be ± 1, grow quite surprisingly. [9, Theorem 1] shows that, for infinitely many n,

$$\log ||\Phi_n||_\infty > \exp\left(\frac{(\log 2)(\log n)}{\log\log n}\right), \tag{2}$$

and indeed this is precisely the right order of (worst-case) growth [10].

When we talk about the difficulties posed by cyclotomic problems we include those posed by "scaled cyclotomics" such as

$$x^{105} - 2^{105} = 2^{105} C_{105}(x/2) :$$

see [11] for more details.

II. The Types of challenges

Ther are several challenges we may meet when trying to devise algorithms for sparse polynomials.

A. It's not sparse?

There are three different ways in which we can have sparse input but nevertheless ask a question where we can't expect the computation time to be polynomial in d_f and t_f.

1) The 'dumb' kind, where we ask a simple question but use a method with an intermediate computation whose answer we have no reason to believe is sparse, such as division with remainder. For example, if one checks whether g divides f via

```
(q,r):=quorem(f,g)
return (r=0)
```

a huge amount of information (all of q and r) is computed and immediately thrown out. The eventual output is obviously sparse (it's a single bit!) Here the only answer is not to ask the question. In particular, the paradigm of computing some extremely complex quantity just to check if (part of) it is $= 0$ is definitely to be avoided. Here some kind of efficient trial division should be used instead.

2) The 'degenerate' kind, where in fact we have encoded a different kind of problem in our polynomial problem. A trivial example is given in section V-A: asking for an integer r such that $f = h^r$ reduces, in the case where $f = x^n$, to enumerating all the factors of n, and in particular to factoring n. Similarly, giving the degrees of the factors of $x^n - 1$ is tantamount to factoring n.
3) The 'exceptional' kind where we would normally expect the results to be sparse, but occasionally they are not. Polynomial factorization furnishes with the best examples. Various sporadic cases are known, e.g. a trinomial of degree 14 with two dense factors of degree 7 [12]. However, the only *known* families of cases of this sort are based around the cyclotomic polynomials. The fundamental issue here is that a cyclotomic polynomial Φ_n, while concise to write *as such*, has degree $\phi(n)$, and generally $\phi(n)$ terms, which may have significantly larger coefficients than one might expect. Elsewhere [11], we argue that the correct solution to this difficulty is *not* to write out Φ_n as a sparse polynomial, but rather to admit either it or C_n as an element of our vocabulary in the first place. In other words, we argue that a further change in representation is needed.

The first case could be called a *software engineering* challenge, where we need to recognize that we're using a sledgehammer to solve a simple problem. Sometimes the 'degenerate' case falls into the same category, when we know in advance the shape of our input. This kind of challenge is frequently encountered in the context of linear algebra, where general LU decomposition routines are all too frequently called with real symmetric positive-definite matrices. Whether to put a front-end routine on routines such as `factor` that 'catch' degenerate cases and route them to more appropriate algorithms is still an open software engineering challenge.

The third case brings us to our first mathematical/algorithmic challenge:

Challenge 1: Find useful bounds on the number of terms in *non-cyclotomic* factors of sparse polynomials.

A special case of this is given as Challenge 4 later.

B. The problem may be intrinsically hard

Most of the classic results in this are are due to Plaisted [13]–[15], as in the following result.

Theorem 1: It is NP-hard to determine whether two sparse polynomials (in the standard encoding) have a non-trivial common divisor.

> The basic device of the proofs is to encode the NP-complete problem of 3-satisfiability so that a formula W in n Boolean variables goes to a sparse polynomial $p_M(W)$ which vanishes exactly at certain Mth roots of unity corresponding to the satisfiable assignments to the formula W, where M is the product of the first n primes. [MR 85j:68043]

Hence this difficulty *as demonstrated* arises with polynomials whose roots are roots of unity, i.e. products of cyclotomic polynomials, when rendered in the standard encoding.

Challenge 2: Either

- find a class of problems for which the gcd problem is still NP-complete even when cyclotomic factors are encoded in one of the encodings (C_n or Φ_k) from [11]; or
- find an algorithm for the gcd of polynomials with *no* cyclotomic factors, which is polynomial-time in the standard encoding.

C. We still don't know how to solve the problem

This is almost universally true — see for example the second clause in challenge 2. The rest of the paper will mostly deal with such cases.

III. Division

Multiplication of sparse polynomials is a relatively well-understood task: the result of $h = f \times g$ has $d_h = d_f + d_g$ and $t_h \leq t_f t_g$, so h may be less sparse than f or g, but not arbitrarily so. In general there are $O(t_f t_g)$ coefficient multiplications to perform. Whereas a naïve algorithm for performing the multiplication may take time $O(t_f^2 t_g)$, with the running time being dominated by sorting the terms of the result, we can achieve $O(t_f t_g \log t_f)$, by better sorting [3].

A. Division with remainder

In the case of division, the obvious "long division" method for f/g requires $\Omega(d_f t_g)$ coefficient operations to be performed. We could write this as $\Omega(t_{f/g} t_g)$, but the absence of a useful answer to Challenge 1 makes this complexity bound only of retrospective use. There are two further issues which complicate the full complexity analysis, which is not often performed.

- The cost of merging the terms (the cost of *not* merging the terms is repeated coefficient operations as two different terms with the same exponent have to be handled). The naïve algorithm is very bad, being $O(d_f^2 t_g)$. Maintaining the partial dividend as a balanced tree gives

$O(d_f t_g \log d_f)$ exponent comparisons. We could also look at the "geobucket" data structure [16].

- Coefficient growth. Unlike multiplication, where the only multiplicative operations are between inputs, there is a feedback process here, which can account for an extra factor of $d_f(\log|f| + \log|g|)$. If one genuinely wants division with remainder, this coefficient growth is intrinsic.

Note how it is always d_f which appears in the above, and not t_f. This should be especially surprising as f appears multiplicatively in f/g.

It appears that Monagan and Pearce [17] have some interesting improvements here, with the total cost of computing f/g with quotient q in $O(t_q t_g \log(\min(t_q, t_g)))$.

B. Exact Division

A more interesting question is that of exact division, where we want the quotient only, or an indication that the division is not exact. In this case, as explained in [18], we can use bounds on the coefficients of factors of f to perform an "early exit" once the coefficients in the putative quotient exceed the Landau–Mignotte bound. Theoretically, this reduces the extra factor mentioned above from $d_f(\log|f| + \log|g|)$ to d_f, but the practical importance seems to be far greater in many cases.

In the standard model, dependence on d_f is inevitable for the reason given in II-A: consider

$$\frac{x^n - 1}{x - 1} = \underbrace{x^{n-1} + \cdots + x + 1}_{n \text{ terms}}. \tag{3}$$

In the models of [11], this would be either

$$\frac{C_n(x)}{C_1(x)} = C_n(x)C_1(x)^{-1} \tag{3–C}$$

or

$$\frac{\prod_{d|n} \Phi_d(x)}{\Phi_1(x)} = \prod_{d|n; d>1} \Phi_d(x). \tag{3–Φ}$$

In neither case is the output significantly larger (in fact, any larger) than the input. A positive answer to Challenge 1 is fundamental here if we want to generalize this.

C. Pure divisibility

If we are purely interested in the question of *whether* g divides f, irrespective of the quotient, and if d_g is small, we can compute f modulo g, using repeated squaring to evaluate $x^n \pmod{g(x)}$, which takes $O\left(t_f \log(d_f)(d_g^2)\right)$ coefficient operations.

It is not clear what to do when d_g is not small. Plaisted has a negative result in this direction, but it doesn't quite answer the question since $\prod_{j=1}^k p_j(x)$ (see below) might not be sparse.

Theorem 2 ([13, Theorem 2.3]): The following problem is NP-hard: given an integer N and a set $\{p_1(x), \ldots, p_k(x)\}$ of sparse polynomials with integer coefficients, to determine whether $x^N - 1$ divides $\prod_{j=1}^k p_j(x)$.

Again, the proof is based on 3-SAT.

Challenge 3: Either

- find a class of problems for which the simple problem "does g divide f?" is still NP-complete; or
- find an algorithm for the divisibility of polynomials which is polynomial-time.

Failing this

- find an algorithm for the divisibility of cyclotomic-free polynomials which is polynomial-time.

IV. Greatest Common Divisor

Theorem 1 shows the difficulties that can arise, at least in the case of cyclotomic polynomials. Furthermore, the following elegant example of [19] shows that the difficulty is real.

$$\gcd(x^{pq} - 1, x^{p+q} - x^p - x^q + 1) = x^{p+q-1} - x^{p+q-2} \pm \cdots - 1 \tag{4}$$

In the models of [11], this would be either

$$\gcd(C_{pq}(x), C_p(x)C_q(x)) = C_p(x)C_q(x)C_1(x)^{-1} \tag{4–C}$$

or

$$\gcd(\Phi_1(x)\Phi_p(x)\Phi_q(x)\Phi_{pq}(x), \Phi_1^2(x)\Phi_p(x)\Phi_q(x)) = \Phi_1(x)\Phi_p(x)\Phi_q(x) \tag{4–Φ}$$

(the last formulation requires p and q to be prime).

As a special case of Challenge 1 we can ask the following.

Challenge 4: Find useful bounds on the number of terms in the greatest common divisor of sparse polynomials in the non-cyclotomic case.

As observed in [20], their Theorem B implies such a bound in terms of t_f, $|f|$, t_g and $|g|$ only (i.e. not involving d_f, d_g), but the bound is not made explicit, and seems to be exponential in t_f and t_g.

By analogy with the algorithms of [3], [17], we can also pose the following.

Challenge 5: Find an algorithm for computing $\gcd(f, g)$ which is polynomial-time in t_f, t_g *and* $t_{\gcd(f,g)}$.

V. Square-free decomposition

We all know that square-free decomposition *can* be computed by g.c.d. computation, but in fact the two problems are equivalent.

Theorem 3 ([21]): Over $\mathbf{Z}$ and in the standard encoding, the two problems

1) deciding if a polynomial is square-free
2) deciding if two polynomials have a non-trivial g.c.d.

are equivalent under randomized polynomial-time reduction.

Hence, in the light of Theorem 1, determining square-freeness is hard, at least when polynomials with cyclotomic factors are involved.

A fortiori, computing the square-free decomposition is hard, at least when cyclotomics are involved. This is certainly the case if we want a full decomposition in the standard model, as the trivial example of

$$x^{p+1} - x^p - x + 1 = (x - 1)^2(x^{p-1} + \cdots + 1) \tag{5}$$

shows. In the models of [11], this would be either

$$C_p(x)C_1(x) = C_1(x)^2\left[C_p(x)C_1(x)^{-1}\right] \tag{5–C}$$

(where $\left[C_p(x)C_1(x)^{-1}\right]$ is a legitimate square-free polynomial) or

$$\Phi_p(x)\Phi_1(x)^2 = \Phi_p(x)\Phi_1(x)^2. \qquad (5\text{–}\Phi)$$

In neither case is the output significantly larger than the input.

We should note that the low-degree part of the square-free decomposition *can* be computed — see Theorem 4 below.

Challenge 6: Find a polynomial-time algorithm for computing the *shape* of the square-free factorization of a polynomial. This might be:

either just the non-trivial multiplicities, e.g. "there are (non-trivial) factors/roots of multiplicities 1, 4 and 7";

or the degrees as well as the multiplicities, e.g. "$f = f_1 f_4^4 f_7^7$ where f_1 has degree 10, f_4 degree 2 and f_7 degree 1.

Equally, we might ask for a polynomial-time algorithm which applies to cyclotomic-free polynomials,

A. Perfect Powers

However, a positive result for the standard representation in this area is provided by [22], [23], who give a Las Vegas polynomial-time algorithm for determining *whether* a given sparse f (not of the form x^n, else the number of possibilities is potentially vast) is h^r, and r itself.

One obvious question is whether h has to be sparse if f is. They conjecture that it does: more precisely the following.

Conjecture 1 ([22, Conjecture 3.1]): For $r, s \in \mathbf{N}$ and $h \in \mathbf{Z}[z]$ with $d_h = s$, then $\hat{t}_{h^i} < \hat{t}_{h^r} + r$ for $1 \leq i < n$, where $\hat{t}_f = t_{f \pmod{x^{2s}}}$.

Assuming this conjecture, they can recover h in polynomial time.

VI. Factorization

Here again, the cyclotomic polynomials immediately lead to problems: consider the factorization of $x^{105} - 1$ as

$$(x-1)\left(x^2+\cdots\right)\left(x^4+\cdots\right)\left(x^6+\cdots\right)\left(x^8 \pm \cdots\right)\left(x^{12} \pm \cdots\right) \times \left(x^{24} \pm \cdots\right)\left(x^{48} \pm \cdots - 2\,x^7 - x^6 - x^5 + x^2 + x + 1\right) \qquad (6)$$

In the models of [11], this would be either $C_{105}(x) =$

$$C_1(x)\frac{C_3(x)}{C_1(x)}\frac{C_5(x)}{C_1(x)}\frac{C_7(x)}{C_1(x)}\frac{C_{15}(x)C_1(x)}{C_3(x)C_5(x)}\frac{C_{21}(x)C_1(x)}{C_3(x)C_7(x)} \times \frac{C_{35}(x)C_1(x)}{C_5(x)C_7(x)}\frac{C_{105}(x)C_7(x)C_5(x)C_3(x)}{C_{35}(x)C_{21}(x)C_{15}(x)C_1(x)} \qquad (6\text{–C})$$

(where each quotient is a legitimate irreducible polynomial) or

$$\Phi_1(x)\Phi_3(x)\Phi_5(x)\Phi_7(x)\Phi_{15}(x)\Phi_{21}(x)\Phi_{35}(x)\Phi_{105}(x) = \text{same} \qquad (6\text{–}\Phi)$$

In fact, one could even write this even more succinctly for larger n as

$$\prod_{\substack{k=1 \\ \gcd(k,n)=1}}^{n} \Phi_k(x).$$

It should be clear that it is really just the divisors of n which matter.

In the light of these examples, and the result of [12] that a trinomial can have factors with eight terms, it might seem hopeless to search for any results on the factorization of sparse polynomials. However, we can find low-degree factors.

Theorem 4 ([24, p. 268]): There is a deterministic algorithm that, for some positive real number c, has the following property: given an algebraic number field K, a sparsely represented non-zero polunomial $f \in K[x]$ and a positive integer d, the algorithm finds all monic irreducible factors of f in $K[x]$ of degree at most d, as well as their multiplicities, and it spends time at most $(l+d)^c$, where l denotes the length of the input data (i.e. $t_f \log(d_f|f|)$).

Challenge 7: Understand the complexity of this result in practice. In particular, we would like to know the value of c in the special case when K is $\mathbf{Q}$.

VII. Other problems

There are many other problems one could pose about sparse polynomials. One about which a little is known is that of polynomial composition, i.e. is $f(x) = g(h(x))$? The case $g(x) = x^d$ is that of perfect powers (see section V-A). Here the first surprising result is the following.

Theorem 5 ([25, Theorem 1]): If h is not of the form $ax^n + b$, then $d_g \leq 2t_f(t_f - 1)$.

This has very recently led to the following, effective in theory but not in practice, result.

Theorem 6 ([26, Theorem 1]): There exists a computable function $\mathcal{B}$ such that if $g, h \in \mathbf{C}[x]$ are non-constant polynomials with $f(x) = g(h(x))$, then $t_h \leq \mathcal{B}(t_f)$.

In other words, if f is of high degree, but has few terms, then g cannot be of high degree (and therefore implicitly has comparatively few terms) and h has few terms. However, these bounds still allow for a surprising degree of cancellation in $f(x) = g(h(x))$. *Some* cancellation is certainly possible as Coppersmith and Danvenport [27] show.

Challenge 8: Understand the complexity of [26, Theorem 1].

VIII. Excluding cyclotomics

Given the difficulties that cyclotomic polynomials seem to cause, we might wish to exclude them, as in Challenges 1, 2, 3 (third point) and 4. However, even this is difficult.

Theorem 7 ([13, Theorem 6.1]): It is NP-hard to solve the problem, given a polynomial $p(x) \in \mathbf{Z}[x]$, to determine if p has a root r of modulus 1.

Of course, we could still *assume* that our input polynomials were cyclotomic-free, since this theorem says nothing about the possibility of it being easy to show the absence of roots of unity, or at least to do so most of the time.

Theorem 7 should also be contrasted with the results below.

Theorem 8 ([28, Theorem 1]): There is an algorithm for determining whether f has a cyclotomic factor which runs in

time

$$\exp\left((2+o(1))\sqrt{t_f/\log t_f}(\log t_f+\log\log d_f)\right)\log(|f|+1) \tag{7}$$

In the case that a cyclotomic factor exists, the algorithm can be made to output a positive integer m for which $\Phi_m(x)$ divides $f(x)$.

Unpicked, (7) says that, for *fixed* t_f, the algorithm is polynomial in $\log d_f$, but the degree of that polynomial depends on t_f.

Subsequently, [20] significantly improves on this by giving an algorithm which will in fact identify the factor of f which has largest degree and composed only of cyclotomic polynomials. Furthermore, once they can rule out cyclotomic factors and other reciprocal factors (i.e. polynomials such that $g(x) = \pm\tilde{g}(x)$ where $\tilde{g}(x) = x^{d_g}g(1/x)$), they produce a complete factorization of such polynomials with running time polynomial in d_f. Unfortunately, it appears that the algorithm is exponential in t_f; it is not even clear what the dependence of the running time on t_f of the last step (factoring of polynomials which contain neither cyclotomic nor reciprocal factors) really is.

We should also note the following result.

Theorem 9 ([29, Theorem 1]): There is an algorithm to decide if $f(\zeta_m) = 0$, whose running time is polynomial in $\log|f|$, $\log d_f$, t_f and $\log m$, i.e. the input size.

IX. Conclusion

The first conclusion is that, as a community, we know comparatively little about sparse polynomial algorithms, despite insisting on their importance. Part of the reason for this is that the cyclotomic polynomials *definitely* cloud the scene, either by letting us produce mathematically trivial but computationally difficult problems, such as (3), (5) or (6), or by their use in deep results such as Theorem 1. Allowing cyclotomics in the vocabulary, as suggested in [11], does something for the first set of problems.

It seems likely that, from the theoretical point of view, we will have to *assume* that the inputs do not have cyclotomic factors if we are to make progress with many of the other challenges, for which Challenge 1 is critical.

Acknowledgment

Much of this work was done while the authors were on sabbatical at the David S. Cheriton School of Computer Science at the University of Waterloo, and the authors would like to thank the Symbolic Computation Group, especially Mark Giesbrecht, for many fruitful discussions.

References

[1] J. Davenport, Y. Siret, and E. Tournier, "Computer Algebra (2nd ed.)," *Academic Press*, 1993.

[2] J. von zur Gathen and J. Gerhard, "Modern Computer Algebra," *C.U.P.*, 1999.

[3] S. Johnson, "Sparse Polynomial Arithmetic," in *Proceedings EUROSAM 74*, 1974, pp. 63–71.

[4] S. Watt, "Making computer algebra more symbolic," in *Proceedings Transgressive Computing 2006*, J.-G. Dumas, Ed., 2006, pp. 43–49.

[5] ——, "Two Families of Algorithms for Symbolic Polynomials," in *Proceedings Computer Algebra 2006: Latest Advances in Symbolic Algorithms*, I. Kotsireas and E. Zima, Eds., 2007, pp. 193–210.

[6] ——, "Functional Decomposition of Symbolic Polynomials," in *Proceedings International Conference on Computational Sciences and its Applications*, 2008, pp. 353–362.

[7] A. Mahboubi, "Proving formally the implementation of an efficient gcd algorithm for polynomials," in *IJCAR*, ser. Lecture Notes in Computer Science, U. Furbach and N. Shankar, Eds., vol. 4130. Springer, 2006, pp. 438–452.

[8] A. Díaz and E. Kaltofen, "FOXBOX: a system for manipulating symbolic objects in black box representation," in *Proceedings ISSAC '98*, O.Gloor, Ed., 1998, pp. 30–37.

[9] R. Vaughan, "Bounds for the Coefficients of Cyclotomic Polynomials," *Michigan Math. J.*, vol. 21, pp. 289–295, 1974.

[10] P. Bateman, "Note on the coefficients of the cyclotomic polynomial," *Bull. AMS*, vol. 55, pp. 1180–1181, 1949.

[11] J. Carette and J. Davenport, "Factorization of Sparse Polynomials or The Power of Vocabulary," *In preparation*, 2009.

[12] A. Bremner, "On Reducibility of Trinomials," *Glasgow Math. J.*, vol. 22, pp. 155–156, 1981.

[13] D. Plaisted, "Sparse Complex Polynomials and Irreducibility," *J. Comp. Syst. Sci.*, vol. 14, pp. 210–221, 1977.

[14] ——, "Some Polynomial and Integer Divisibility Problems are NP-Hard," *SIAM J. Comp.*, vol. 7, pp. 458–464, 1978.

[15] ——, "New NP-Hard and NP-Complete Polynomial and Integer Divisibility Problems," *Theor. Comp. Sci.*, vol. 31, pp. 125–138, 1984.

[16] T. Yan, "The geobucket data structure for polynomials," *J. Symbolic Comp.*, vol. 25, pp. 285–294, 1998.

[17] M. Monagan and R. Pearce, "Sparse Polynomial Division Using a Heap," http://www.cecm.sfu.ca/CAG/papers/MonHeaps.pdf, 2009.

[18] J. Abbott, R. Bradford, and J. Davenport, "A Remark on Factorisation," *SIGSAM Bulletin 2*, vol. 19, pp. 31–33, 1985.

[19] A. Schinzel, "On the greatest common divisor of two univariate polynomials, I," *A Panorama of number theory or the view from Baker's garden*, pp. 337–352, 2003.

[20] M. Filaseta, A. Granville, and A. Schinzel, "Irreducibility and greatest common divisor algorithms for sparse polynomials," *Number Theory and Polynomials*, pp. 155–176, 2008.

[21] M. Karpinski and I. Shparlinski, "On the Computational Hardness of Testing Square-Freeness of Sparse Polynomials," in *Proceedings AAECC-13*, M. Fossorier, H. Imai, S. Lin, and A. Poli, Eds., 1999, pp. 492–497.

[22] M. Giesbrecht and D. Roche, "On Lacunary Polynomial Perfect Powers," in *Proceedings ISSAC 2008*, D.J.Jeffrey, Ed., 2008, pp. 103–110.

[23] ——, "Detecting lacunary perfect powers and computing their roots," http://arxiv.org/abs/0901.1848, 2009.

[24] H. Lenstra Jr., "Finding small degree factors of lacunary polynomials," *Number theory in progress*, pp. 267–276, 1999.

[25] U. Zannier, "On the number of terms of a composite polynomial," *Acta Arith.*, vol. 127, pp. 157–167, 2007.

[26] ——, "On composite lacunary polynomials and the proof of a conjecture of Schinzel," *Invent. Math.*, vol. 174, pp. 127–138, 2008.

[27] D. Coppersmith and J. Davenport, "Polynomials whose Powers are Sparse," *Acta Arithmetica*, vol. 58, pp. 79–87, 1991.

[28] M. Filaseta and A. Schinzel, "On testing the divisibility of lacunary polynomials by cyclotomic polynomials," *Math. Comp.*, vol. 73, pp. 957–965, 2004.

[29] S. Tarasov and M. Vyalyi, "An Efficient Algorithm for Zero-Testing of a Lacunary Polynomial at the Roots of Unity," in *Proceedings Computer Science - Theory and Applications*, ??, pp. 397–406.