CM730/CS687: Introduction to Symbolic Computation Final Report: Polynomial-time Algorithm for Super-sparse Polynomials with Integer Coefficients

Ce Ju (20632264)

Computational Math Program

April 19, 2017

Contents

1 Introduction

In this survey, we present the history of the algorithms of super-sparse (lacunary) polynomials including the complexity issue and recent research and the main idea of the breakthrough polynomial-time algorithm from Cucker-Koiran-Smale. In the end, we implement their algorithm in *Maple*.

2 Super-sparse Polynomial

In the field of algebraic number theory, we call a polynomial over a finite fields $\mathbb{F}$ **fully reducible** if it factors completely into linear factors over $\mathbb{F}$. According to the classical definition from L*á*szl*ó* R*é*dei, the **lacunary** polynomial is defined as a fully reducible polynomial with the gap between the highest degree and the second highest degree no more than two. Intuitively, a lacunary polynomial is the polynomial with a long sequence of zero coefficients. In computer algebra, the lacunary polynomial is called **(super)-sparse** polynomial. But, some poineers like A.Schinzel and H.W.Lenstra, Jr still call them the lacunary polynomial. In the following paragraph, the super-sparse polynomial is defined in a multivariate form.

A super-sparse polynomial

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{k} a_i x_1^{\beta_{i,1}} \cdots x_n^{\beta_{i,n}},$$

where all coefficients a_i are integers.

The size of polynomial f is defined as

$$\text{size}(f) = \sum_{i=1}^{k} \big(\text{size}(a_i) + \lceil \log_2(\beta_{i,1} \cdots \beta_{i,n} + 2) \rceil\big).$$

In Turing model of computation and complexity, we say an algorithm of the polynomial is solvable in **polynomial time** meaning that the the number of bit operation to complete the algorithm is bounded by $\mathcal{O}(\text{size}(f)^n)$ for some nonnegative number n.

3 History of Super-sparse Polynomial Algorithms

3.1 Computational Complexity and Super-sparse Polynomial

In computational complexity, we call a question which can be answered by *Yes* or *No* as a **decision** question. A class of questions in all decision questions are called **NP** class if the *Yes* instance of each question can be verified by a deterministic algorithm in polynomial time. Suppose we have two decision problems P_1 and P_2. We say that P_1 **polynomial-time reduces** to P_2 if every instance of P_1 can be solved by polynomial number of computational steps and calls of P_2 as a subroutine at unite cost. And, if we can construct any *Yes* instance of P_2 by the polynomial number of costs of *Yes* instance of P_1, we say P_1 **polynomial-time transforms** to P_2. If all problems in class NP can polynomial-time reduce to some problem but we are unable to know if this problem is belongs to NP or other class, we call this class of problems as **NP-hard**. And, If all problems in class NP can polynomial-time transform to some problem in class NP, we call this class of problems as **NP-complete**.

The original study of sparse polynomial algorithms dates back to D.A.Plaisted in 1977 and 1984. In his paper ([Pla77, 1977]), several problems related to the sparse polynomials with integer coefficients are NP-hard via construction the polynomial-time reduction from **tautology detection problem**. A polynomial-time algorithm to solve P1 to P6 will prove the P versus NP problem eventually. The following table records these problems.

Problem	Description of Problems
P1	Given a finite sequence of polynomials, determine
	The lcm. The degree of the lcm. Whether the lcm is equal to some polynomial. Whether the degree of lcm is equal to some number.
P2	If a product of finitely many polynomials has some specific polynomial as a factor.
P3	How many distinct zero roots of a product of polynomials.
P4	If the number of distinct zero roots is equal to some number.
P5	If $\frac{p(x)}{q(x)}$ is analytic, where $p(x)$ is a product of finitely many polynomials and $q(x)$ is a polynomial.
P6	$\text{GCD}(f(x), x^n - 1) = 1$

In his following paper ([Pla84, 1984]), he continues to show that several polynomials are NP-hard or even NP-complete. The following tables recodes these group of problems.

Class	Problem
NP-complete	Sparse-poly-divis
NP	Sparse-poly-nonroot
NP-hard	Sparse-poly-root-modulus-1 2-Sparse-poly-gcd Trig-inequality Bounded recurrence Sparse matrix eigenvalue Common-diff-eq-solution Sparse-poly-quotient Sparse-poly-remainder 0,1-Sparse-poly-nonfactor Sparse-binary-divis Ring-homomorphism injection

The description of these problems can be found in his paper. Since knowing them is far from the goal of this report, we skip all the descriptions.

3.2 Recent Research of Super-sparse Polynomial Algorithms

The breakthrough algorithm ([CKS98, 1998]) is what we will mainly present in this report (See Section 4). They design a polynomial-time algorithm to compute the integer roots of the univariate super-sparse polynomial $f \in \mathbb{Z}(x)$. H.W.Lenstra, Jr ([LJ99a, LJ99b, 1999]) generalizes their results and design a new algorithm to compute the lower degree factors of the univariate super-sparse polynomial over an algebraic extension in polynomial time. In ISSAC' 05 and 06, E.Kaltofen and P.Koiran design an algorithm to compute the linear factors of bivariate polynomials over $\mathbb{Q}$ ([KK05, 2005]) and then low degree factors of multivariate polynomials over an algebraic extension ([KK06, 2006]). The following table records the general input and output of these four algorithms.

Cucker-Koiran-Smale' 1998:

- Input: Super-sparse $f \in \mathbb{Z}(x)$.
- Output: A set of integer roots of f.

Lenstra' 1999: Suppose a monic irreducible polynomial $\varphi(\zeta) \in \mathbb{Z}(\zeta)$ and the algebraic number field $\mathbb{K} := \mathbb{Q}(\zeta)/\varphi(\zeta)$.

- Input: Factor degree d and super-sparse $f \in \mathbb{K}(x)$.
- Output: A set of all irreducible factors of f with the degree no more than d and there multiplicities.

Kaltofen-Koiran' 2005:

- Input: A bivariate super-sparse polynomial $f(x_1, x_2) = \sum_{i=1}^{k} a_i \cdot x_1^{\beta_{i,1}} x_2^{\beta_{i,2}} \in \mathbb{Z}(x_1, x_2)$, which is monic in x_1 and the error probability ϵ.
- Output: A set of factors of f named as $g_i(x_1, x_2)$ with both degrees with respect to x and y no more than 2 and their multiplicities. The factors $g_i(x_1, x_2)$ are irreducible of f over $\mathbb{Q}$ with the possibilities over $1 - \epsilon$.

Kaltofen-Koiran' 2006: Suppose a monic irreducible polynomial $\varphi(\zeta) \in \mathbb{Z}(\zeta)$ and the algebraic number field $\mathbb{K} := \mathbb{Q}(\zeta)/\varphi(\zeta)$.

- Input: Super-sparse $f(x_1, \ldots, x_n) = \sum_{i=1}^{k} a_i x_1^{\beta_{i,1}} \cdots x_n^{\beta_{i,n}} \in \mathbb{K}(x_1, \ldots, x_n)$ and factor degree bound d.
- Output: A set of all irreducible factors of f over $\mathbb{K}$ with the degree no more than d and there multiplicities.

4 A Breakthrough Algorithm from Cucker-Koiran-Smale' 1998

A breakthrough polynomial-time algorithm is from F.Cucker, P.Koiran and S.Smale in 1998 ([CKS98, 1998]). The description of the problem is presented in Section 3.2 but we restate it again, i.e. there is an algorithm which given a (super)-sparse function with integer coefficients can output the integer roots in a polynomial time of size f. Fortunately, the algorithm is constructive and thus we implement the algorithm in *Maple*. (See Section 7)

The main technique of their algorithm is called **Gap Theorem** in the literature. Suppose an univariate super-sparse polynomial $f(x) = \sum_{i=0}^{n} a_i \cdot x^{\beta_i}$. We can write $f(x)$ into two parts, i.e. $f(x) = g(x) + x^{\beta_{k+1}} \cdot h(x)$, where $g(x) = \sum_{i=0}^{k} a_i \cdot x^{\beta_i}$ and $h(x) = \sum_{i=k+1}^{n} a_i \cdot x^{\beta_i - \beta_k}$. Keep in mind that our polynomial is super-sparse and the degree of $f(x)$ is super large. It means that $g(x)$ and $x^{\beta_{k+1}} \cdot h(x)$ may have a big gap between the degree β_k and β_{k+1} for some k. The amazing technique is from the following theorem. If the gap between coefficients $a_k - a_{k+1}$ is over some bound, the integer roots x ($|x| \geq 2$) of $f(x)$ is also the roots of $g(x)$ and $h(x)$. We can pick the bound as $\log_2 \sum_{i=0}^{n} |a_i|$, but there is a sharp bound $1 + \log_2 \sup_{0 \leq i \leq k} |a_i|$ in their paper. Actually, the Gap Theorem from Cucker-Koiran-Smale is a variant of Cauchy's bound of the roots of a polynomial.

Cauchy's Bound: Given a monic polynomial $f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Then, all zero roots x of $f(x)$ satisfies the following inequality

$$|x| \leq 1 + \max_{i=0,\cdots,n-1} |a_i|.$$

For the implement part, an easy try is to check if $\frac{\beta_n}{n} > 1 + \log_2 \sup_{0 \leq i \leq n} |a_i|$ for the polynomial $f(x)$ with a nonzero constant term a_0. If it does, we can always pick the gap index k. Then, we compute the integer roots of two small size polynomial $g(x)$ and $h(x)$ and get their common integer roots.

In this paragraph, we talk about the algorithm how to compute the integer roots of a small size polynomial in polynomial time from F.Cucker, P.Koiran and S.Smale. The idea is to collect a set of finitely covers of all possible integer roots of $f(x)$ and then check them one by one. It grows up from Rolle's theorem and part of Strum's theorem. Actually, Strum's algorithm works well for the case of the dense polynomials but not well for the sparse case. For example, our polynomial is $x^2 - x - 6$ and we start from the initial cover $\mathcal{C}_0 = \{[0, 0]\}$. Note that all the covers $[u, v]$ ($u \leq v$) are length 0 or 1. Since every integer root can divide constant term a_0, we let the whole interval be as $[-6, 6]$. The first procedure is to compute all order derivatives of $f(x)$ up to only constant left. We name them as p_1, p_2, p_3 as follows,

$$\begin{aligned} p_1 &:= x^2 - x - 6, \\ p_2 &:= 2x - 1, \\ p_3 &:= 2. \end{aligned}$$

Suppose we have a finitely disjoint cover $\mathcal{C} := \{[u_i, v_i]\}_{i=1}$ of all the possible integer roots of the derivative $f'(x)$, then each open interval (v_i, u_{i+1}) may have at most one root for $f(x)$. If there are more than one root in this interval, there is a root between (v_i, u_{i+1}) by Rolle's theorem and thus it contradicts with our assumption of the construction of the cover $\mathcal{C}$. The way to detect if there is a root in (v_i, u_{i+1}) is to compute the signedness on the boundary point of the intervals. If the signedness are opposite, it means there is one root. There is a polynomial-time algorithm from Cucker-Koiran-Smale for this procedure and we will mention it a little later. Thus, if we want to have the finitely cover $\mathcal{C}$ of all the possible integer roots of the derivative $f(x)$, we need the finitely cover $\mathcal{C}$ of all the possible integer roots of the derivative $f'(x)$ and then we need all the possible integer roots of the derivative $f''(x)$ until of all order derivative $f^{(n)}(x)$.

Back to our example, $\mathcal{C}_0 := \{[0, 0]\}$ is a cover of p_3 since the derivative of p_3 is 0. Then, we construct the cover $\mathcal{C}_1$ by taking the complement of $\mathcal{C}_0' = \{(-6, 0), (0, 6)\}$. Check the signedness on $(-6, 0, 6)$, i.e. compute the sign of $p_2(x)$ on these points and get the sign $(-, -, +)$. Since $p_2(-6) \cdot p_2(0) > 0$ and $p_2(0) \cdot p_2(6) < 0$, there is no root between $(-6, 0)$ for p_2 but there is one between $(0, 6)$. We continue to narrow this interval to $(0, 1)$ by bisection method on integer point and get the cover $\mathcal{C}_1 := \{[0, 0], [0, 1]\}$. For construction of the cover $\mathcal{C}_2$, we utilize the same routine as the construction of $\mathcal{C}_1$. Take the complement of $\mathcal{C}_1$ and get $\mathcal{C}_1' = \{(-6, 0), (1, 6)\}$. Then, check the signedness on $(-6, 0, 1, 6)$ by p_1 and then get the sign $(+, -, -, +)$. It means both intervals have a root and narrow to small size interval by bisection method and get the cover $\mathcal{C}_2 = \{[-3, -2], [0, 0], [0, 1], [3, 3]\}$.

The last procedure of this algorithm is to find the exact roots by computing the exact value from the set of the boundary points of the intervals in the cover $\mathcal{C}_2$. It is obviously that only -2 and 3 are the true roots of $x^2 - x - 6$. The only left part of this algorithm is the polynomial-time sign algorithm. The idea is from Horner's method but there is a big difference from our ordinary way since we only need the signedness rather than the exact value. Actually, Horner's method cannot be achieved in polynomial time and thus the authors design an algorithm to compute the rough approximation of the exact value but it is enough to know the signedness. Firstly, they compute a rough bound for the innermost layer $a_n \cdot x_0 + a_{n-1}$ and then pass the bound through the layers out. In each loop i, it updates the signedness of the function constructed by the inner i layer at value x_0 until the last loop. The algorithm is based on several basic procedures which are called the *straight-line* program and proved to be polynomial-time. Thus, the sign algorithm is polynomial-time and so is the whole algorithm from Cucker-Koiran-Smale.

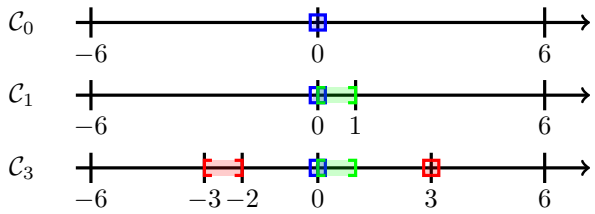

Figure 1: Polynomial $f(x) = x^2 - x - 6$ with derivatives $p_1 = f(x), p_2 = 2x - 1$ and $p_3 = 2$. $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2$ are the covers of the possible roots of p_3, p_2 and p_1 respectively. The blue is the initial cover, the green is added after the first loop and the reds are added after the second loop.

5 Numerical Experiment

The algorithm from Cucker-Koiran-Smale is implemented in *Maple*. We coded 13 procedures of *Maple* including Self_coeffs, Lem_1, Step_1, Step_2, Sign_Main, Bisection, Prop_1, TurnBracket, Dev, Thm_1_Main, Self_*nops*, CommonRoot and Prop_2. The relations between these 13 procedures are shown in the following forrest diagram. The child node will call their parent nodes as a subroutine. The main procedures are Thm_1_Main and Prop_2. Thm_1_Main implement the Theorem 1 and Prop_2 implement the Proposition 2 on the paper [CKS98, 1998]. Proposition 2 applies the gap technique and speed up the algorithm of Theorem 1.

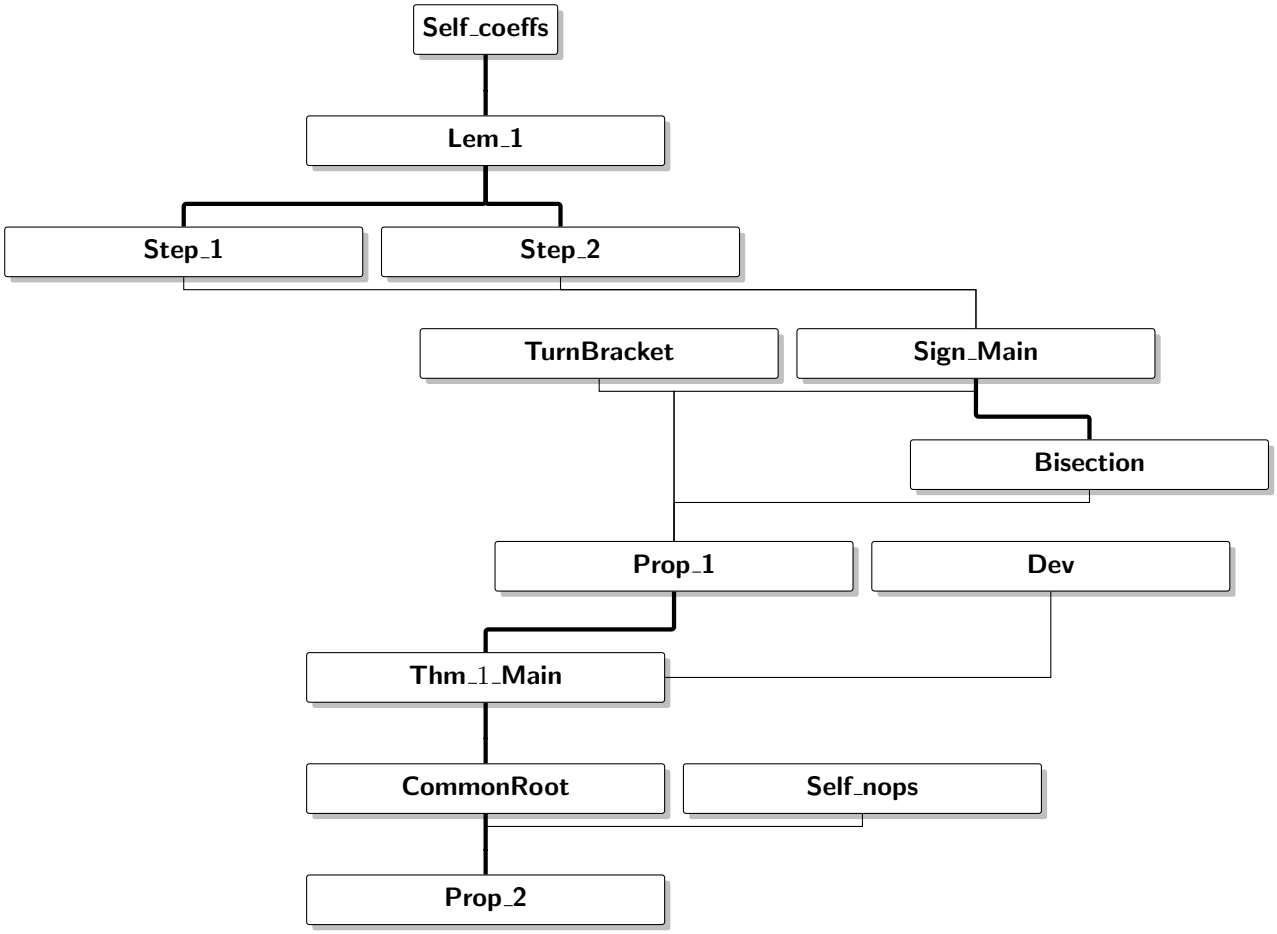

Figure 2: Forest Diagram of Procedures in My Code

For testing our implement, Let's try the example in Section 4 firstly.

```
>>Prop_2(x^2-x-6);
               {-2, 3}
```

Correct! In the following part, we will try a sequence of examples of super-sparse polynomials.

```
>>Prop_2(x^9-92x^8-3538x^7+147908x^6+1230741x^5-4671720x^4+3296700x^3);
               {-45, -10, 0, 1, 2, 33, 111}
>>Prop_2(x^1999-2x+1)
               { }
>>Prop_2(x^2000-x^1908+x-1)
               {1}
```

Then, let's test how faster the algorithms between Prop_2, Thm_1_Main and the build-in function *roots* of *Maple*. In Figure 3, we plot the time assumption of three algorithms to compute the integer roots of sparse polynomials. The x-axis represents the degrees of the sparse polynomials from 10, 100, 1000, 10,000 to 100,000, but we narrow them by $\log_{10}$. Each time value is an average of 50 numerical experiments of random polynomials. The blue line represents Prop_2 and the red represents Thm_1_Main. From the figure, the blue one is much lower than the red one after the degree is growing up. It means that the gap technique make a big contributions in the algorithm. In Figure 4, we also plot the time assumption of three algorithms to compute the integer roots of dense polynomials rather than sparse. The x-axis represents the degrees of the sparse polynomials from 5, 15, 25, 35 to 45. We pick small degrees on purpose. Since the algorithms takes much time for the dense polynomial case. From the figure, we notice Prop_2 and Thm_1_Main performs equivalently. Actually, the gap technique won't work in most of the dense case. The yellow one always representing the build-in function is just a control group.

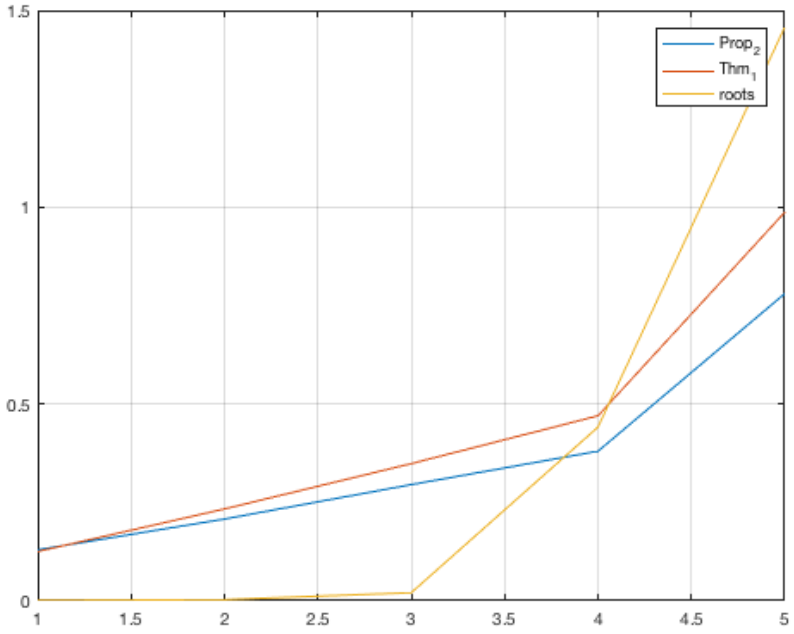

Figure 3: Comparison of the Time Assumption of Three Algorithms for Sparse Polynomials

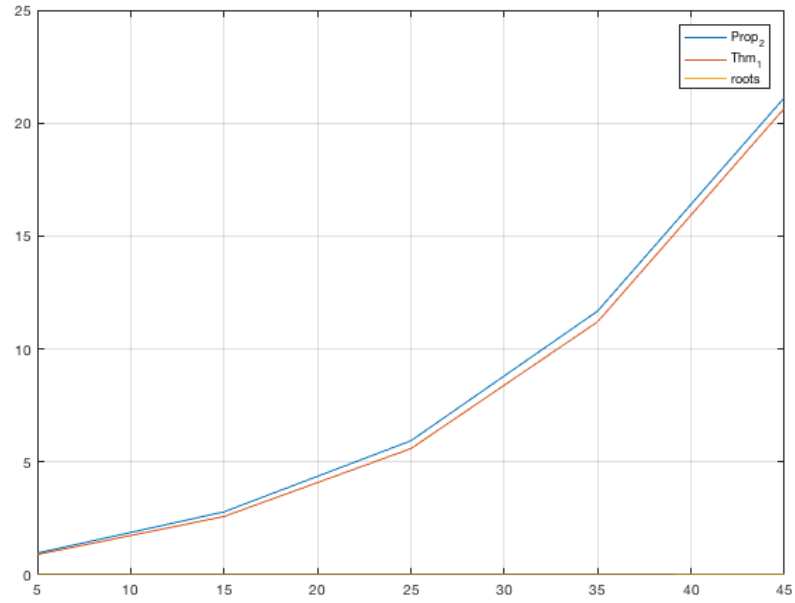

Figure 4: Comparison of the Time Assumption of Three Algorithms for Dense Polynomials

6 Conclusion

The super-sparse algorithm is a very charming topic whatever in research or in practice. Although there are bunches of problems are NP-hard, the algorithm from F.Cucker, P.Koiran and S.Smale makes a breakthrough contribution in this field so that people start to realize the existence of polynomial-time algorithm for some simple problem in the related field. And then, the scholars make a series of the improvements after their algorithm. In the near future, why don't we try to solve them in light of quantum computer?

References

[CKS98] Felipe Cucker, Pascal Koiran, and Steve Smale. A polynomial time algorithm for diophantine equations in one variable. *Journal of Symbolic Computation*, 27(1):21–29, 1998.

[KK05] Erich Kaltofen and Pascal Koiran. On the complexity of factoring bivariate supersparse (lacunary) polynomials. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 208–215. ACM, 2005.

[KK06] Erich Kaltofen and Pascal Koiran. Finding small degree factors of multivariate supersparse (lacunary) polynomials over algebraic number fields. In *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 162–168. ACM, 2006.

[LJ99a] Hendrik W Lenstra Jr. Finding small degree factors of lacunary polynomials. *Number theory in progress*, 1:267–276, 1999.

[LJ99b] Hendrik W Lenstra Jr. On the factorization of lacunary polynomials. *Number theory in progress*, 1:277–291, 1999.

[Pla77] David Alan Plaisted. Sparse complex polynomials and polynomial reducibility. *Journal of Computer and System Sciences*, 14(2):210–221, 1977.

[Pla84] David A Plaisted. New np-hard and np-complete polynomial and integer divisibility problems. *Theoretical Computer Science*, 31(1-2):125–138, 1984.

[PS82] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.