Faster Algorithms for Sparse Decomposition and Sparse Series Solutions to Differential Equations

by

Saiyue Lyu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Sparse polynomials are those polynomials with only a few non-zero coefficients relative to their degree. They can appear in practice in polynomial systems as inputs, where the degree of the input sparse polynomial can be exponentially larger than the bit length of the representation of it. This leads to the difficulties when computing with sparse polynomials, as many efficient algorithms for dense polynomials take polynomial-time in the degree, and hence an exponential number of operations in a natural representation of the sparse polynomial.

In this thesis, we explore new and faster methods for sparse polynomials and power series. We reconsider algorithms for the sparse perfect power problem and derive a faster sparsity-sensitive algorithm. We then show a fast new algorithm for sparse polynomial decomposition, again sensitive to the sparsity of the input and output. Finally, our algorithms to solve the sparse perfect power and decomposition problems lead us to explore a generalization to solving the linear differential equation with sparse polynomial coefficients using a Newton-like method. We demonstrate an algorithm which will find sparse solutions if they exist, in time polynomial in the input and the output.

Acknowledgements

First and foremost, I would like to thank Mark Giesbrecht for being a kind, supportive, inspiring and knowledgeable supervisor. With the guidance of Mark, I learned how to model and define problems, formalize research methodology and practice meticulous academic writings. I struggled a lot over the past two years, Mark has always been lenient and patient with me. I want to thank him for his mentorship to my development as a researcher and a person.

I also want to thank Arne Storjohann for being my supportive co-supervisor. Two years ago, I took a course with Arne and he introduced me to the world of symbolic computation. I thank Arne for his guidance and influence on me. Also, I would like to thank two readers, George Labahn and Robert Corless, for reading this thesis and providing interesting and valuable feedback. I thank Reinhold Burger, Haomin Li and everyone else from the Symbolic Computation Group for their support.

I would like to thank Yu-Ru Liu at Waterloo, with whom I started to do research. I was an undergraduate research assistant with Yu-Ru in Pure Math, and her kindness and encouragement allowed me to tide over the difficulties, finish my bachelor degree and make up my mind for graduate studies. I cannot overstate the warmth she brings to me.

On a more personal note, I would like to thank my lifelong best friend Jiayi Eris Zhang since grade seven. Our discussion about research and life never ends. Thanks for making everything better. I also thank my friends, Muge Yuan, Xuejun Du, Haonan Duan, Chuyi Liu, Jiayuan Li, Yixin Chen, Zhenyuan Zhang, Wanchun Rosie Shen, Jingjing Wu, for being there for me throughout my worst and best time.

I would like to thank my supportive family members: Yinliang Xiao, Xiaojian Lyu, Yuzhen Chai, Jincang Cao, Wei Lyu, Shengping Ding, Sihan Ding, Ning Lyu, Haifeng Zhao, Yihan Zhao, Xiaohong Cao, Wei Yuan, Muge Yuan. I cannot survive the past six years without any of you.

It has been a very different period due to COVID-19, I would like to thank all the health care workers to support our daily life.

Last but not least, I want to thank my parents, Kai Lyu and Xiaoyan Cao, for their unconditional support and love. Thanks for always believing in me and never giving up on me.

Dedication

This is dedicated to my lovely family.

Table of Contents

List of Figures viii

1 Introduction 1

2 Background and Related Work 3

2.1 Sparse Polynomials . . . 3

2.1.1 Computing with Sparse Polynomials . . . 4

2.2 Perfect Power Problem for Sparse Polynomials . . . 6

2.3 Polynomial Decomposition . . . 8

2.4 Differential Equations Over Formal Power Series . . . 9

3 New Algorithms for Sparse Perfect Power and Polynomial Decomposition 12

3.1 Perfect Power Problem for Dense Polynomials . . . 14

3.1.1 When $f_0 \neq 0$ . . . 15

3.1.2 When $f_0 = 0$ . . . 16

3.1.3 A Combinatorial Proof of Equation (3.1) . . . 18

3.2 Algorithms for Sparse Perfect Powers . . . 21

3.2.1 When $f_0 \neq 0$ . . . 21

3.2.2 When $f_0 = 0$ . . . 24

3.3 Perfect Power Problem for the "Wild" Case . . . 25

3.4 Perfect Power Problem for Rational Functions 28
3.5 Sparse Polynomial Decomposition in the "Tame" Case 30

4 Generalization to Differential Equations over Formal Power Series 33
4.1 Undetermined Coefficient Method 34
4.2 The $\ell = 1, 2$ cases 36
4.3 Polynomial differential equations of arbitrary order 39

5 Implementations and Experiments 42
5.1 Implementation and Tests 42
5.2 Experiment Results 44
5.2.1 Structure a 44
5.2.2 Structure b and Structure c 45
5.2.3 Structure d : Perfect Power Problem 46

6 Conclusion and Future Work 48

References 50

List of Figures

3.1 The main structure of Chapter 3 14

5.1 Experiment results for structure a 44
5.2 Experiment results for structure b 45
5.3 Experiment results for structure c 46
5.4 Perfect power experiment result comparison 47

Chapter 1

Introduction

Sparse polynomials are those polynomials with many zero coefficients. They often appear in practice in polynomial systems as inputs, where the specification of a problem through polynomials might have a small number of constraints (as represented by non-zero coefficients) relative to the degree. Mathematically, they can also appear as solutions to polynomial problems due to some structure or constraint, which may or may not be well understood.

The degree of a sparse polynomial can be exponentially larger than the bit length of the representation of it. This leads to the difficulties when computing with sparse polynomials, as many efficient algorithms for dense polynomials (where most coefficients are assumed to be non-zero, or we are oblivious to whether they are zero) take polynomial-time in the degree, and hence and exponential number of operations in a natural representation of the sparse polynomial.

On the mathematical side, there are famous conjectures which have only be recently resolved. The conjecture of Erdös (1949) asks whether a given sparse polynomial can only be a perfect power of similarly sparse polynomial. This has only been recently (partially) resolved by Zannier (2008). Another classic problem is sparse polynomial decomposition: given sparse $f \in F[x]$ with degree $n = rs$, determine whether there exists $g, h \in F[x]$ such that $g \circ h = f$ with deg $g = r$ and deg $h = s$. Mathematically, Zannier (2008) has shown that if f is sparse, then h must be sparse as well, and g must be of relatively low degree. Even here, Zannier (2008) only shows sparsity of h to be computable from the sparsity of f (independent of the degree), and tight bounds are not known.

On the computational side, some fundamental problems like computing the gcd of two sparse polynomials are provably intractable, due to foundational early work of Plaisted

(1977). Other problems, like determining the irreducibility of a univariate sparse polynomial, are unknown. However, there are algorithms for some important related problems. Giesbrecht and Roche (2008, 2011) give a Las Vegas polynomial-time algorithm to determine whether a given sparse polynomial is a perfect power, which is polynomial-time in the input *and the output size*. This output sensitivity makes the solution independent of the mathematical conjectures which bound the output size. We employ this approach of output-size sensitivity in this thesis as well.

In this thesis, we explore new and faster methods for sparse polynomials and power series. We reconsider algorithms for the sparse perfect power problem, and derive a faster algorithm than that presented by Giesbrecht and Roche (2011). We then show a fast new algorithm for sparse polynomial decomposition, again sensitive to the sparsity of the input and output. We believe it is the first provably polynomial-time algorithm known for this problem. Finally, our algorithms to solve the sparse perfect power and decomposition problems lead us to explore a generalization to solving the linear differential equation with sparse polynomial coefficients using a Newton-like method. We demonstrate an algorithm which will find sparse solutions if they exist, in time polynomial in the input and the output.

The remainder of the thesis is structured as follows.

In Chapter 2, we will discuss the basic mathematical concepts for sparse polynomials and the generalized linear differential equation with sparse polynomial coefficients, and the related work for computing with sparse polynomials, especially the algorithms for perfect power problem.

In Chapter 3, we will derive a new algorithm to solve the perfect power problem. Two proofs of the algorithm are presented (an inductive one and a direct combinatorial one). We show how this algorithm can be modified to derive a new algorithm for sparse polynomial decomposition.

In Chapter 4, we generalize the undetermined coefficient method to solve a linear differential equations over power series to exploit sparsity. We derive a Newton-like algorithm to accomplish this.

In Chapter 5, we describe the implementations and experiment results of the presented algorithms for general differential equation and the special case perfect power.

Open questions for future exploration are discussed in Chapter 6.

Chapter 2

Background and Related Work

In this chapter, we will review the background knowledge and previous work of this thesis. Section 2.1 reviews basic concepts and arithmetic for sparse polynomials. Section 2.2 discusses the setup for perfect power problem and the mathematical background for solving it. Section 2.3 introduces the polynomial decomposition problem, which can be inspired by perfect power problem. Section 2.4 reviews basic concepts about differential equations over formal power series and introduces the generalized linear differential equation problem with sparse polynomial coefficients.

2.1 Sparse Polynomials

Sparse polynomials, also called lacunary or supersparse polynomials, are those polynomials with relatively few nonzero terms compared to their degrees. Alternatively, they are polynomials with significant "gaps" or "lakes" (lacuna) between the degreess of consecutive terms. For example $f = x^{100} + x^2$ is sparse. We will refer to standard polynomials as "dense" polynomials, where most terms are non-zero, or at least we treat them as such; for example $f = x^4 + x^3 + x^2 + x + 1$.

Definition 2.1.1 (sparsity, support, sparse representation)**.**

Consider a sparse polynomial $f \in \mathsf{F}[x]$ *of degree* n*,*

$$\begin{aligned} f &= \sum_{i=0}^{n} f_i x^i \quad \textit{where } n = \deg f \textit{ and } f_n \neq 0 \\ &= \sum_{j=1}^{\tau(f)} f_{n_j} x^{n_j} \quad \textit{where } f_{n_j} \neq 0 \textit{ and } n_1 < n_2 < \ldots < n_{\tau(f)} = n. \end{aligned}$$

We call $\tau(f)$ *the* **sparsity** *of* f*,* $n_1, \ldots, n_{\tau(f)}$ *the* **exponents** *of* f*, and* $x^{n_1}, \ldots, x^{n_{\tau(f)}}$ *the* **support** *of* f*.*

The representation of f *is a set of coefficients and exponents, we call it the* **sparse representation** *of* f*, so in particular* f *as above is represented as*

$$\left\{(f_{n_1}, n_1), \ldots, (f_{n_{\tau(f)}}, n_{\tau(f)})\right\},$$

and has size $O(\tau(f) \log n)$ *assuming that field elements are represented with unit size.*

2.1.1 Computing with Sparse Polynomials

We denote τ as the sparsity bound of the input polynomials. Note that n can be exponentially larger than τ, i.e. the degree can be much larger than the bit-length of the representation. This leads to the difficulty when computing with sparse polynomials. Usually researchers have to modify the dense algorithms or come up with new algorithms for sparse polynomials to achieve a complexity which is sensitive to the sparsity. Roche (2018) surveyed recent progress for sparse polynomials algorithms in arithmetic, interpolation and factorization. Sparse **addition** can be solved using a merge operation, which requires $O(\ell^2 \tau \log n)$ when adding/subtracting ℓ sparse polynomials. We can avoid the quadratic-time cost when each polynomial is ordered by the exponents, and addition can be solved using a max-heap in $O(\ell\tau \log n \log \ell) = \tilde{O}(\ell\tau \log n)$ time. For **multiplication**, the output size potentially grows quadratically. Dense polynomials of degree n can be multiplied in $O(n \log n)$ operations Harvey and Van Der Hoeven (2019), but for the sparse case, the classic algorithm of repeated monomial multiplications takes $O(\tau^2 \log n \log \tau)$ operations as discussed in Roche (2018); it is still an open problem to avoid quadratic time.

For sparse division, things become more difficult as the output size can be large even if the input is sparse, for example, $x^n - 1$ divides $x - 1$ produces $1 + x + \cdots + x^{n-1}$. Monagan and Pearce (2011) developed heap-based algorithms to compute the quotient q

for given sparse polynomials f, g such that $f = qg + r$ in $O(\tau(q)\tau(g)\log(\min(\tau(q), \tau(g))))$, which is still quadratic in τ, it is also an open problem to avoid quadratic time to achieve $\tilde{O}(\tau \log n)$ operations. Even the corresponding decision problem is not known to have a polynomial-time algorithm:

Problem 2.1.2 (sparse polynomial exact division decision)**.** *Given two sparse polynomials f_1, f_2, the maximum degree of these polynomials is n and the maximum sparsity of f_1, f_2 is τ, determine whether f_2 divides f_1.*

As discussed in Plaisted (1984), it is not known that whether Problem 2.1.2 is in **P** or whether it is **NP**-complete, though Grigoriev et al. (1992) proved that determining the nondivisibility of sparse polynomials is in **NP** if the Extended Riemann Hypothesis holds, i.e., Problem 2.1.2 belongs to the class **co-NP**. Plaisted (1977, 1984); Davenport and Carette (2009) also pointed out that some other basic computing problems involving sparse polynomials are **NP**-hard although there already exists fast algorithms for these problems when the input is dense.

Problem 2.1.3 (gcd of 2 sparse polynomials over integer field)**.** *Given two sparse polynomials $f_1, f_2 \in \mathbb{Z}[x]$ with* **integer** *coefficients, determine whether f_1 and f_2 are not relatively prime, i.e.* $\deg(\gcd(f_1, f_2)) > 0$.

Theorem 2.1.4 (Theorem 5.1 in Plaisted (1984))**.** *Problem 2.1.3 is* **NP***-hard. Proof is a reduction based on cyclotonmic polynomials.*

Also note that the computation of GCD relates to square-free decomposition.

Definition 2.1.5 (square-free polynomial)**.** *A polynomial f is said to be square-free if f has no divisor (factor) of multiplicity ≥ 2.*

Problem 2.1.6 (square-freeness)**.** *Given a sparse polynomial $f \in \mathbb{Z}[x]$ with* **integer** *coefficients, determine whether f is square-free.*

If α is a repeated root of f, then it is a root of f', and thus a root of $\gcd(f, f')$. Also it is known that the roots of $\gcd(f, f')$ are exactly the repeated roots of f. So a square-free polynomial has no repeated roots, thus $\gcd(f, f')$ has no nontrivial roots. Using randomized poly-time reduction, Karpinski and Shparlinski (1999) proved the the equivalence of square-freeness and gcd :

Theorem 2.1.7 (Theorem 1 & 2 in Karpinski and Shparlinski (1999))**.** *Problem 2.1.3 and Problem 2.1.6 are equivalent, both are* **NP***-hard. The results also hold over the algebraic closure of a finite field with prime characteristic.*

2.2 Perfect Power Problem for Sparse Polynomials

We now consider the perfect power problem defined as :

Problem 2.2.1 (Polynomial perfect power)**.** *Given* $f = \sum_{i=0}^{n} f_i x^i \in \mathsf{F}[x]$ *with* $\deg f = n$ *and* $f_n \neq 0$*, suppose that* $n = rs$ *for some* $r, s \in \mathbb{N}$*, determine whether there exists* $h \in \mathsf{F}[x]$ *with* $\deg h = s$ *such that* $f = h^r$*.*

Steady progress for developing fast sparse polynomial algorithms has been proceeded over the past few decades, especially for the sparse perfect power problem :

Problem 2.2.2 (Sparse polynomial perfect power)**.** *Given sparse* f *of degree* n*, suppose* $n = rs$ *for some* $r \in \mathbb{N}$*, determine if there exists some* $h \in \mathsf{F}[x]$ *of degree* s *such that* $f = h^r$*. If so, find* h *in a manner whose cost is polynomial in* $\log n, \tau(f), \tau(h)$*.*

To test whether f is a perfect power, for dense polynomials, it is efficient to check whether each exponent of the squarefree decomposition of f is divisible by 2, which is discussed by Yun (1976). Other methods also exist, and effectively require linear time in the degree of the input. For sparse polynomials, Giesbrecht and Roche (2008, 2011) give a Las Vegas poly-time algorithm to determine whether a given sparse polynomial f is h^r.

To find the polynomial root h, things can be more difficult. Using Newton iteration, Brent and Kung (1978); von zur Gathen (1990) showed that computing the root h takes $O(M(n) \log r)$ operations if $f(0) = 1$ and r not divisible by $\mathrm{char}(F)$, where $M(n) = n \log n \log \log n$. For the the sparse case, the first question is whether h is sparse? Erdös's famous conjecture asks, when f is sparse and $f = h^2$, whether h is "kind of" sparse:

Conjecture 2.2.3 (Erdös (1949))**.** *Let* $h \in \mathsf{F}[x]$ *be a polynomial of* $\tau(h)$ *non-zero terms, consider* $f = h^2$*. Let* $\tau(f)$ *denote the sparsity of* h^2*, define* $Q(\tau(h)) = \min\{\tau(f) : f = h^2$ *where* h *is a sparse polynomial with* $\tau(h)$ *terms*$\}$*, then there exists constants* $0 < c_1 < 1$ *and* $0 < c_2$ *such that* $Q(\tau(h)) < c_2 \cdot \tau(h)^{1-c_1}$*.*

Later Schinzel (1987); Zannier and Schinzel (2009) generalized the result of square to any power:

Theorem 2.2.4 (Theorem 1 & 2 in Zannier and Schinzel (2009))**.** *Let* $h \in \mathsf{F}[x]$ *and* $f = h^r$ *with* $\tau(h) \geq 2$*,*

1) if either $char(\mathsf{F}) = 0$ *or* $char(\mathsf{F}) > r \cdot \deg h$*, then*

$$\tau(f) \geq 2 + \frac{\log(\tau(h) - 1)}{\log 4r};$$

2) if $char(\mathsf{F}) > 0$ *and* $r^{\tau(h)-1}(\tau(h)^2 - \tau(h) + 2) < char(\mathsf{F})$*, then*

$$\tau(f) \geq 2 + \frac{\log(\tau(h) - 1)}{\log 4r}.$$

Schinzel (1987) conjectured the case when g is a fixed perfect power polynomial, then Zannier (2007) proved the result for any general g that a polynomial with few terms and large degree cannot have an inner noncyclic composition factor of small degree:

Theorem 2.2.5 (Theorem 1 in Zannier (2007) & Theorem 2 in Zannier (2008))**.** *Let* $f \in \mathsf{F}[x]$ *with* $\tau(f)$ *sparsity. Suppose* $f = g \circ h$ *for some* $g, h \in \mathsf{F}[x]$ *and* h *is not of the form* $ax^s + b$*. Then*

$$\begin{aligned} \deg g &\leq 2\tau(f)(\tau(f) - 1) < 2\tau(f)^2, \\ \deg f + \tau(f) - 1 &\leq 2\tau(f)(\tau(f) - 1) \cdot \deg h. \end{aligned}$$

In his breakthrough partial proof of Schinzel's conjecture, Zannier (2008) showed that the degree of g is related to the sparsity of f instead of the degree of f, and that $\tau(h)$ is a computable function of $\tau(f)$, independent of the degree, i.e., that "sparse" polynomials have sparse decompositions. It is expected that $\tau(h)$ is comparable to $\tau(f)$, though this is not known. This also means if $f = h^r$, then $r < 2\tau(f)^2$.

The perfect power problem is a special case of polynomial decomposition, where $g = x^r$. With the work by Erdös, Schinezel and Zannier, and all known examples supporting this, Giesbrecht and Roche (2008, 2011) made the following conjecture:

Conjecture 2.2.6 (Conjecture 3.1 in Giesbrecht and Roche (2008))**.** *For* $r, x \in \mathbb{N}$*, if* $char(F) = 0$ *or* $> rs$*, and* $h \in F[x]$ *with degree* s*, then*

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

Assuming this conjecture, Giesbrecht and Roche (2008, 2011) presented an algorithm to compute h using a kind of Newton iteration in polynomial time.

A private observation by Koiran (2011) suggests a faster algorithm for sparse perfect power is possible as follows. Assume $f = h^r$. Then

$$f' = r \cdot h^{r-1}h' = r \cdot (f/h)h'$$

giving

$$f'h = rfh', \tag{2.1}$$

where f' and h' are the usual derivatives of f and h respectively. That is, h is a kind of solution to a differential equation involving f and f'. Note that this assumes the characteristic of F does not divide r, or in other words the image of r in F is non-zero. We will assume this throughout this thesis. We explore this approach in more detail in Chapter 3.

One follow-up problem of Problem 2.2.2 is to consider the perfect power problem for rational functions:

Problem 2.2.7 (rational function perfect square)**.** *Given a rational function f of the form $f = f_1/f_2$ for some sparse polynomials $f_1, f_2 \in F[x], f_2 \neq 0$ of degrees n_1, n_2 respectively, determine if there exists some rational $h = h_1/h_2$ with $h_1, h_2 \in F[x], h_1, h_2 \neq 0$ such that $f = h^r$, if so, find h in a manner whose costs is poly$(\tau(f_1), \tau(f_2), \tau(h_1), \tau(h_2), \log n_1, \log n_2)$.*

A discussion of this problem is presented in Section 3.4.

2.3 Polynomial Decomposition

Let F be any field and $g, h \in \mathsf{F}[x]$, we say $f = g \circ h = g(h(x)) \in \mathsf{F}[x]$ is the composition of g and h, and the pair (g, h) is a decomposition of f. The polynomial decomposition problem is:

Problem 2.3.1 (polynomial decomposition)**.** *Given $f = \sum_{i=0}^{n} f_i x^i \in \mathsf{F}[x]$ with $\deg f = n$ and $f_n \neq 0$, suppose that $n = rs$ for some $r, x \in \mathbb{N}$, determine whether there exists $g, h \in \mathsf{F}[x]$ with $\deg g = r$ and $\deg h = s$ such that $f = g \circ h$.*

Note that the composition of given polynomials is unique but the decomposition of a given polynomial may not be unique. A polynomial may have distinct decompositions: $f = g_1 \circ g_2 \circ \cdots \circ g_m = h_1 \circ h_2 \circ \cdots \circ h_n$, where $g_i \neq h_i$ for some i. If the degrees of the polynomials are given and we fix the constant term, Ritt (1922) proved that $m = n$ and the degrees of the polynomials in one decomposition are the same as those in the other, but possibly in different order. For this thesis, we only consider to find one of the decompositions of prescribed degree, if it exists, and the problem will be normalized so that this solution is unique.

For dense polynomials, Barton and Zippel (1985) gave an exponential-time algorithm by directly solving $rs + 1$ equations obtained by equating the coefficients of f and $g(h)$. After this, Kozen and Landau (1989) presented an $O(n^2)$-time algorithm without using

polynomial factorization, which works over any commutative ring containing a multiplicative inverse of r. Based on the work of Kozen & Landau, von zur Gathen (1990) used the trick of reversal to improve the cost to $O(n \log^2 n \log\log n)$ for the "tame" case, where the characteristic of the field does not divide r.

In this thesis, we also assume we are in the "tame" case. Following von zur Gathen (1990), we may assume that f, g, h are monic and $h(0) = 0$. von zur Gathen (1990) uses an observation on polynomial reversals which we will also exploit. Let $\tilde{f}(x) = x^n \cdot f(1/x)$, the *reversal of* f. Similarly, let $\tilde{h}(x) = x^s \cdot h(1/s)$. He then shows that $f(x) = g(h(x))$ implies $\tilde{f}(x) \equiv \tilde{h}(x)^r \bmod x^s$. Moreover, since $\deg \tilde{h}(x) = \deg h = s$ and $h(0) = 0$, if such an $\tilde{h}$ exists it is unique. Thus, von zur Gathen (1990) showed computing h such that there exists a g with $f(x) = g(h(x))$ is equivalent to computing $\tilde{h}$ such that $\tilde{h}^r \equiv \tilde{f} \bmod x^s$. The polynomial g such that $f(x) = g(h(x))$ can be found by interpolation.

For sparse polynomials, it is not at all clear that a similar approach will work, and indeed Giesbrecht and Roche (2011) did not solve this, though did present a conjectural approach which is close. In Section 3.5 we present a faster algorithm for sparse polynomial decomposition inspired by solving our new solution to the sparse perfect power problem.

2.4 Differential Equations Over Formal Power Series

Note that solving the perfect power problem is in fact solving the differential equation (2.1). We would like to consider the general differential equation problem, looking for sparse power series solutions. In this section, we will review the basic concepts about the linear differential equations with polynomial coefficients.

Definition 2.4.1 (the ring of formal power series)**.**

Recall that the expression $\sum_{i\geq 0} a_i x^i$ *for* $a_i \in F$ *is called a* **formal power/Taylor series**, *and* **the ring of formal power series** *over* F, *denoted as* $F[[x]]$, *is defined as the set of expressions of the above form, i.e.*

$$\mathsf{F}[[x]] = \left\{ \sum_{i\geq 0} a_i x^i \quad \textit{for } a_i \in \mathsf{F} \right\},$$

with addition and multiplication defined as:

$$\sum_{i\geq 0} a_i x^i + \sum_{i\geq 0} b_i x^i = \sum_{i\geq 0} (a_i + b_i) x^i, \quad \Big(\sum_{i\geq 0} a_i x^i\Big) \cdot \Big(\sum_{i\geq 0} b_i x^i\Big) = \sum_{i\geq 0} \Big(\sum_{j=0}^{i} a_j b_{i-j}\Big) x^i.$$

This forms a commutative ring.

Definition 2.4.2 (formal derivative, differential operator)**.**

For

$$a = \sum_{i \geq 0} a_i x^i \in \mathsf{F}[[x]],$$

we define the **formal derivative** *as*

$$a' = \sum_{i \geq 0} i a_i x^{n-1} \in \mathsf{F}[[x]],$$

and define the **differential operator** $\mathcal{D} : \mathsf{F}[[x]] \to \mathsf{F}[[x]]$ *such that* $\mathcal{D}(a) = \mathcal{D}a = a'$*. Similarly* $\mathcal{D}^2 a = \mathcal{D}(\mathcal{D}(a))$ *is the second derivative of* a*, and* $D^k a = D(D^{k-1}(a))$ *is the* k*th derivative, etc.*

Note that (2.1) is in fact a differential equation:

$$0 = f'h - rfh' = (f' - rf\mathcal{D})h,$$

where $(f' - rf\mathcal{D})$ is a given linear operator, so finding h is equivalent to solving this differential equation. So solving perfect power problem is a special case of solving the general linear differential equation.

Definition 2.4.3 (linear differential operator (with polynomial coefficients))**.**

A linear differential operator (with polynomial coefficients) *is defined as a polynomial*

$$\mathcal{L} = f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2 + \cdots + f_\ell\mathcal{D}^\ell$$

for $f_0, \ldots, f_\ell \in \mathsf{F}[x]$ *and* $f_\ell \neq 0$*. We call the integer* ℓ *the* **order** *of* $\mathcal{L}$*.*

Such a linear operator can be applied to a formal power series $a \in \mathsf{F}[[x]]$ as above by

$$\mathcal{L}a = \mathcal{L}(a) = f_0 a + f_1\mathcal{D}(a) + f_2\mathcal{D}^2(a) + \cdots + f_\ell\mathcal{D}^\ell(a).$$

Definition 2.4.4 (the ring of linear differential operators with polynomial coefficients)**.**

The ring of linear differential operators with polynomial coefficients *or skew polynomials or Ore polynomials is defined to be the set of all linear differential operators with polynomial coefficients, i.e.*

$$\mathsf{F}[x][\mathcal{D};'] = \left\{ f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2 + \cdots + f_\ell\mathcal{D}^\ell \ : \ f_i \in \mathsf{F}[x] \right\}.$$

The addition is defined by the usual polynomial addition (+) and the multiplication is defined by

$$\mathcal{D}x = x\mathcal{D} + 1.$$

We can quickly check that the above definition gives a ring. If $\mathcal{L}_1, \mathcal{L}_1 \in \mathsf{F}[x][\mathcal{D};']$, then clearly $\mathcal{L}_1 + \mathcal{L}_2 \in \mathsf{F}[x][\mathcal{D};']$ and easily $(\mathcal{L}_1 + \mathcal{L}_2)(a) = \mathcal{L}_1(a) + \mathcal{L}_2(a)$. Straightforward manipulation also shows that that $(\mathcal{D}x)(a) = (x\mathcal{D}+1)(a) = x\mathcal{D}(a)+a = xa'+a$ and more generally. $(\mathcal{D}f(x))(a) = (f\mathcal{D} + f')(a) = f\mathcal{D}(a) + f'a = fa' + f'a$. It is easily shown that $\mathsf{F}[x][\mathcal{D};']$ is a non-commutative ring.

The application of linear differential operators

$$\begin{aligned} \mathsf{F}[x][\mathcal{D};'] \times \mathsf{F}[[x]] &\to \mathsf{F}[[x]] \\ (\mathcal{L}, a) &\mapsto \mathcal{L}(a) \end{aligned}$$

forms a **ring action** of $\mathsf{F}[x][\mathcal{D};']$ on $\mathsf{F}[[x]]$. A **solution** of a linear differential operator $\mathcal{L} \in \mathsf{F}[x][\mathcal{D};']$ is a Taylor series $a \in \mathsf{F}[[x]]$ such that $\mathcal{L}a = 0$.

We now consider the following computational problem for linear differential equations.

Problem 2.4.5 (linear differential equation)**.** *Suppose we are given*

$$\mathcal{L} = f_0 + f_1\mathcal{D} + \cdots + f_\ell\mathcal{D}^\ell \in \mathsf{F}[x][\mathcal{D};'], \tag{2.2}$$

where $\tau(f_i) \le t$ for $0 \le i \le \ell$ (i.e., each f_i has at most t non-zero terms) and an $m \in \mathbb{N}$. The problem is to find an $h \in \mathsf{F}[x]$ such that

$$\mathcal{L}h \equiv 0 \bmod x^m,$$

i.e., that all terms of h of degree less than or equal to m are zero. The polynomial h is an "approximation" modulo x^m to a power series solution to the differential equation. Sometimes we have a priori *knowledge that h must be sparse. Our goal is to design an algorithm which finds such h with $(\ell + t + \tau(h))^{O(1)}$ operations in F, i.e., polynomial in the input and output size.*

Previous work has discussed the m-sparse solution for linear differential equation with polynomial coefficients (Abramov, 2000; Abramov and Ryabenko, 2004). For an m-sparse polynomial $f = \sum_i f_i x^i$, it means there exists an integer N such that $(f_i \neq 0) \Rightarrow (i \equiv N \bmod m)$, i.e. the gaps of the nonzero terms are divided by N. Problem 2.4.5 consider the settings when we consider the sparsity of the polynomials, which was not studied by previous work. More details are discussed in Chapter 4.

Chapter 3

New Algorithms for Sparse Perfect Power and Polynomial Decomposition

In this chapter, we examine two related problems for a sparse polynomial $f \in F[x]$.

- Perfect powers: determining is there is a formal polynomial r-th root $h \in F[x]$ of f, that is, an h such that $f = h^r$, where r is a positive integer, and if so computing h;
- Polynomial decomposition: determining if there exists a polynomial $g \in F[x]$ and $h \in F[x]$ such that $f(x) = g(h(x))$, and if so computing g, h.

For both these problems we will present algorithms whose output is polynomial in the sparsity of both the input f and the output h. Following Zannier (2008) we expect that if f is sparse then h is (reasonably) sparse. Generally we will assume that $\deg f = n$, $\deg g = r$ and $\deg h = s$, so $n = rs$. We will use f_k, g_k, h_k to denote the k- coefficient of f, g, h respectively in this chapter.

This work is predicated on a clever observation of Koiran (2011), relating perfect powers to logarithmic derivatives (this observation is also employed in Giesbrecht and Roche (2011)). In Section 3.1, we discuss two algorithms to compute the r-th root h of f, which are derived by differentiating both sides of $f = h^r$. One algorithm assumes f_0 is nonzero and recovers the coefficients of h in the order from h_0 to h_s, the other deal with the case when $f_0 = 0$ and recovers the coefficients of h in the order from h_s to h_0. We observe that the cost of each algorithm is in terms of the degree s, which can be a problem when we move to the sparse setting. We also give a combinatorial proof of the algorithm.

In Section 3.2, to solve the perfect power problem for sparse polynomials, we develop fast algorithm modified from the dense case using a Newton-like way to update the residual in each iteration and find the next nonzero exponent of h. The cost of the algorithm is in terms of the sparsity of f and the sparsity of h as desired, and is faster than those previously known (Giesbrecht and Roche, 2011).

Our algorithms in Section 3.2 require that characteristic of the field F not divide the power r. This is analogous to polynomial decomposition where the "wild" case (characteristic divides the degree) is much harder than the "tame" case. See von zur Gathen (1990). In Section 3.3, we discuss how to solve the perfect power problem for sparse polynomials problem in the "wild" case, by reducing it to the "tame" case.

In Section 3.4, we discuss the challenges when taking the rational functions into consideration and address interesting open problems.

Finally, in Section 3.5 we examine the problem of sparse polynomial decomposition in the "tame" case. We show that a reduction of von zur Gathen (1990) is sensitive to sparsity, and transforms the problem into a variant of perfect powers we can solve with our methods. This is the first polynomial-time algorithm for sparse polynomial decomposition.

A pictorial representation of how some of the ideas in this chapter can be derived is illustrated in Figure 3.1. We will fill in the details in the subsequent sections of the chapter.

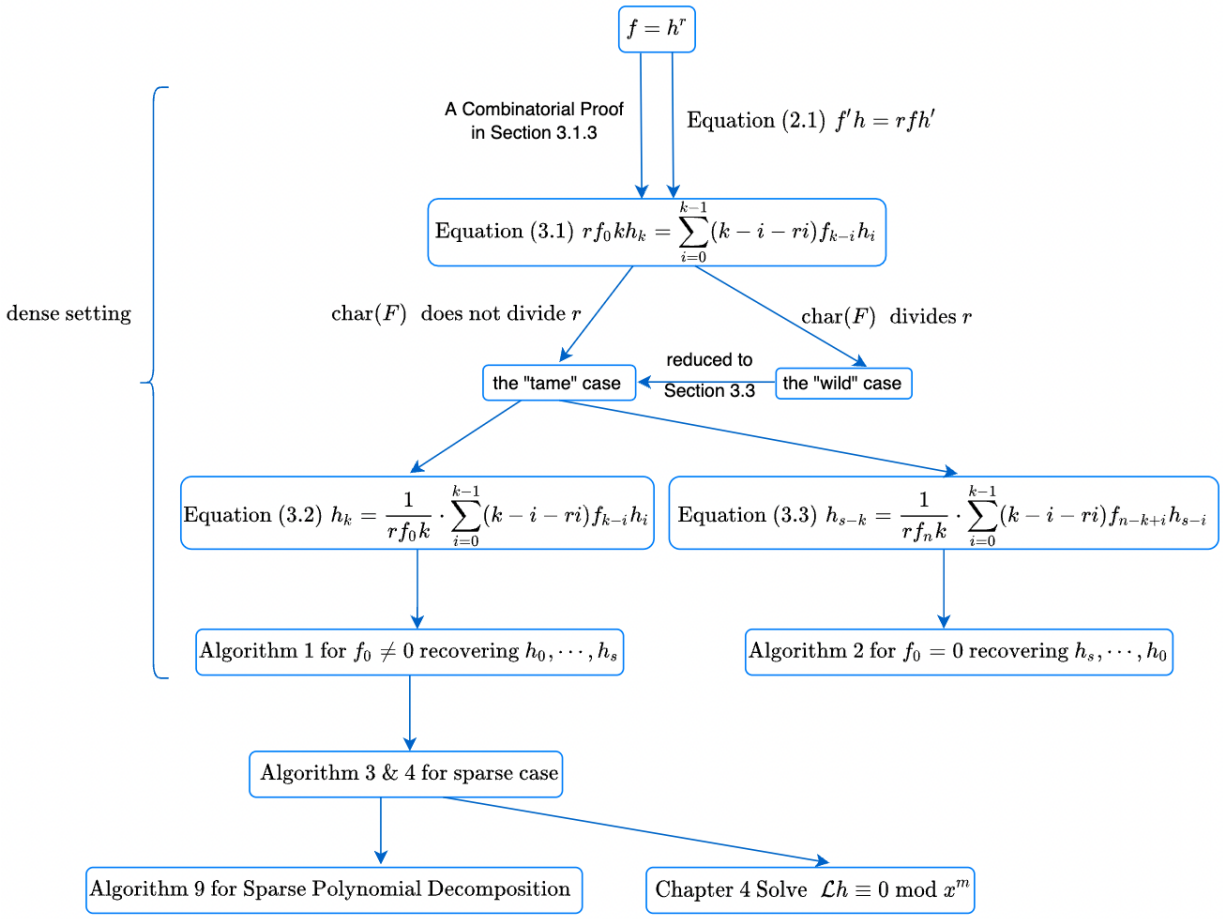


Figure 3.1: The main structure of Chapter 3

3.1 Perfect Power Problem for Dense Polynomials

In this section we develop a recurrence relation for the coefficients of the polynomial rth root of another polynomial. It is presented here for dense polynomials and later adapted to sparse polynomials.

As before, we assume that $f \in F[x]$ is a polynomial of degree n, $h \in F[x]$ is a polynomial of degree s, $f = h^r$ and $n = rs$. For what follows we will not worry about whether f is sparse or dense. Following the standard notation of generating functions, we write $[x^k]f = f_k$ for

the kth coefficient of f and $[x^k]h = h_k$ for the kth coefficient of h . We observe that

$$[x^k]f' = (k+1)[x^{k+1}]f = (x+1)f_{k+1}.$$

To solve Problem 2.2.1, we look at both sides of Equation (2.1) $f'h = rfh'$:

$$[x^k](f'h) = \sum_{i=0}^{k}(k+1-i)f_{k+1-i}h_i,$$
$$[x^k](rfh') = r\sum_{i=0}^{k}(i+1)f_{k-i}h_{i+1} = \sum_{i=1}^{k+1}rif_{k+1-i}h_i.$$

Thus for all $k \geq 0$, we have

$$\begin{aligned}
[x^k](rfh') &= [x^k](f'h),\\
\sum_{i=1}^{k+1}rif_{k+1-i}h_i &= \sum_{i=0}^{k}(k+1-i)f_{k+1-i}h_i,\\
\sum_{i=0}^{k+1}rif_{k+1-i}h_i &= \sum_{i=0}^{k}(k+1-i)f_{k+1-i}h_i, \qquad \text{change the index to } i=0,\\
\sum_{i=0}^{k}rif_{k+1-i}h_i + r(k+1)f_0h_{k+1} &= \sum_{i=0}^{k}(k+1-i)f_{k+1-i}h_i,\\
rf_0(k+1)h_{k+1} &= \sum_{i=0}^{k}(k+1-i-ri)f_{k+1-i}h_i.
\end{aligned}$$

Shifting k down by 1 and we will get, for all $k \geq 1$,

$$rf_0kh_k = \sum_{i=0}^{k-1}(k-i-ri)f_{k-i}h_i. \tag{3.1}$$

3.1.1 When $f_0 \neq 0$

Note that when $f_0 \neq 0$, (3.1) will give:

$$h_k = \frac{1}{rf_0k}\cdot\sum_{i=0}^{k-1}(k-i-ri)f_{k-i}h_i. \tag{3.2}$$

Equation (3.2) says that given coefficients of f of degree up to k and coefficients of h up to degree $k-1$, we can determine h_k. We have to compute h_0 separately, but clearly $h_0 = f_0^{1/r}$. Note that f_0 has r different rth roots, and it is conventional to choose a real root (e.g., $(-8)^{1/3} = -2$) but this is not necessary. This suggests an algorithm to solve Problem 2.2.1 perfect power problem for dense polynomials:

Algorithm 1: DensePerfectPowerNonzeroConst

Input: polynomial $f \in \mathsf{F}[x]$ of degree n s.t. $f_0 \neq 0$ and integer $s, r \in \mathbb{Z}$ s.t. $n = rs$
; **Output:** polynomial $h \in \mathsf{F}[x]$ such that $f = h^r$
; $h_0 = f_0^{1/r}$;
$h = h_0$;
for $k = 1, \ldots, s$ **do**
  $temp = 0$;
  for $i = 0 \cdots k-1$ **do**
    $temp+ = (k - i - ri) \cdot \text{coeff}(h, i) \cdot \text{coeff}(f, k - i)$;
  end
  $h_k = \frac{1}{rf_0 k} \cdot temp$;
  $h+ = h_k \cdot x^k$;
end
return h;

Theorem 3.1.1. *Algorithm 1 takes $O(s^2)$ operations. The justification is straightforward.*

Note that when we consider the sparse case, i.e. when f and h are sparse, Algorithm 1 takes $O(s \cdot \tau(f))$ operations, which can be quite large when s, the degree of h, is large. We need to construct an algorithm which is sensitive to $\tau(f)$ and $\tau(h)$.

3.1.2 When $f_0 = 0$

When $f_0 = 0$, shifting k up by 1 in (3.1) will give:

$$\sum_{i=0}^{k} (k + 1 - i - ri) f_{k+1-i} h_i = 0, \forall 1 \leq k \leq n + s - 1.$$

We substitute index k by $n + s - 1 - k$ to obtain:

$$\sum_{i=0}^{n+s-1-k} (n + s - 1 - k + 1 - i - ri) f_{n+s-k-i} h_i = 0.$$

Note that when $i > s, h_i = 0$, when $i < s - k, f_{n+s-k-i} = 0$ as $n + s - k - i > n$, thus:

$$\sum_{i=s-k}^{s} (n + s - 1 - k + 1 - i - ri) f_{n+s-k-i} h_i = 0.$$

Shifting i down by $s - k$ gives:

$$\sum_{i=0}^{k} (n + s - 1 - k + 1 - (s - i) - r(s - i)) f_{n-k+i} h_{s-i} = 0,$$

$$rk f_n h_{s-k} + \sum_{i=0}^{k-1} (ri + i - k) f_{n-k+i} h_{s-i} = 0.$$

Note $f_n \neq 0$ as the leading coefficient, so it gives:

$$h_{s-k} = \frac{1}{r f_n k} \sum_{i=0}^{k-1} (k - i - ri) f_{n-k+i} h_{s-i}. \tag{3.3}$$

Equation (3.3) says that given coefficients of f of degree from $n - k$ to $n - 1$ and coefficients of h of degree from $s - k + 1$ to s, we can determine h_{s-k}. This is different from the $f_0 \neq 0$ case as we are recovering h_k from high degree to low degree. Also we have to compute h_s separately, but clearly $h_s = f_n^{1/r}$. This suggests the following algorithm to solve Problem 2.2.1 perfect power problem for dense polynomials:

Algorithm 2: DensePerfectPowerZeroConst

Input: polynomial $f \in \mathsf{F}[x]$ of degree n s.t. $f_0 = 0$ and integer $s, r \in \mathbb{Z}$ s.t. $n = rs$
Output: polynomial $h \in \mathsf{F}[x]$ such that $f = h^r$

```
h_s = f_n^{1/r};
h = h_s · x^s;
for k = 1 ··· s − 1 do
    temp = 0;
    for i = 0 ··· k − 1 do
        temp+ = (k − i − ri) · coeff(h, s − i) · coeff(f, n − k + i);
    end
    h_{s−k} = 1/(r f_n k) · temp;
    h+ = h_{s−k} · x^{s−k};
end
return h;
```

Theorem 3.1.2. *Similarly as before, Algorithm 2 takes $O(s^2)$ operations.*

Still, we need to come up with a sparsity-sensitive algorithm to solve Problem 2.2.2 when $f_0 = 0$.

3.1.3 A Combinatorial Proof of Equation (3.1)

In this subsection we present an alternative combinatorial proof of Equation (3.1), to get a further understanding of the underlying mathematical structure. Before we jump into the sparse case, note that Algorithm 1 and Algorithm 2 are both based on (3.1), which is also:

$$\begin{aligned} rf_0h_k &= \sum_{i=0}^{k-1} \frac{k-i-ri}{k} f_{k-i}h_i \\ &= \sum_{i=0}^{k-1} f_{k-i}h_i - \frac{r+1}{k}\sum_{i=0}^{k-1} i \cdot f_{k-i}h_i. \end{aligned} \tag{3.4}$$

This has a strong combinatorial meaning as the subscripts of f_{k-i} and h_i sum up to a fixed number k. So we now try to prove Equation (3.4) without using differentiation.

Note that $fh = h^{r+1}$, we have:

$$[x^k](h^{r+1}) = \sum_{i=0}^{k} f_{k-i}h_i = \sum_{i=0}^{k-1} f_{k-i}h_i - f_0h_k. \tag{3.5}$$

Also note that $[x^k](h^{r+1})$ must have the term $(r+1)h_0^r h_k = (r+1)f_0h_k$, say:

$$[x^k](h^{r+1}) = (r+1)f_0h_k + M, \tag{3.6}$$

for some M being a sum of some coefficient products. Combining (3.5) and (3.6), we have:

$$\begin{aligned} (r+1)f_0h_k + M &= \sum_{i=0}^{k-1} f_{k-i}h_i - f_0h_k \\ rf_0h_k &= \sum_{i=0}^{k-1} f_{k-i}h_i - M. \end{aligned}$$

So we need to figure out what is M. Consider:

$$\begin{aligned} [x^k](h^{r+1}) = [x^k](\sum_{k=0}^{s} h_k x^k)^{r+1} &= [x^k] \sum_{k=0}^{s(r+1)} \Big(\sum_{\substack{0\le j_1,\cdots,j_{r+1}\le k \\ j_1+\cdots+j_{r+1}=k}} h_{j_1}\cdots h_{j_{r+1}} \Big) x^k \\ &= \sum_{\substack{0\le j_1,\cdots,j_{r+1}\le k \\ j_1+\cdots+j_{r+1}=k}} h_{j_1}\cdots h_{j_{r+1}}. \end{aligned} \tag{3.7}$$

Similarly we can have:

$$[x^k](f) = [x^k](h^r) = \sum_{\substack{0\le t_1,\cdots,t_r\le k\\ t_1+\cdots+t_r=k}} h_{t_1}\cdots h_{t_r},$$
$$f_{k-i} = [x^{k-i}](f) = [x^{k-i}](h^r) = \sum_{\substack{0\le t_1,\cdots,t_r\le k-i\\ t_1+\cdots+t_r=k-i}} h_{t_1}\cdots h_{t_r},\ \forall i = 0,\cdots,k. \tag{3.8}$$

Now we modify (3.7) a little bit. Consider for any choice of $j_1,\cdots,j_{r+1}$ satisfying the running condition:

$$0 \le j_1,\cdots,j_{r+1} \le k, \qquad j_1+\cdots+j_{r+1} = k.$$

Say there are $\sigma \le r+1$ distinct elements such that $\{p_1,\cdots,p_\sigma\} = \{j_1,\cdots,j_{r+1}\}$, and each p_i appears α_i times in $\{j_1,\cdots,j_{r+1}\}$, i.e.:

$$p_1\cdot\alpha_1+\cdots+p_\sigma\cdot\alpha_\sigma = j_1+\cdots+j_{r+1} = k.$$

Thus we have:

$$\begin{aligned}[x^k](h^{r+1}) &= \sum_{\substack{0\le j_1,\cdots,j_{r+1}\le k\\ j_1+\cdots+j_{r+1}=k}} h_{j_1}\cdots h_{j_{r+1}}\\ &= \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}\le k\\ j_1+\cdots+j_{r+1}=k}} \binom{r+1}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}}.\end{aligned} \tag{3.9}$$

where $\binom{r+1}{\alpha_1,\cdots,\alpha_\sigma} = \frac{(r+1)!}{\alpha_1!\cdots\alpha_\sigma!}$ is a multinomial coefficient. Note that $(\alpha_1,\cdots,\alpha_\sigma)$ depends on $(j_1,\cdots,j_{r+1})$. We can do similar rearrangement for Equation (3.8) to have:

$$\begin{aligned}f_{k-i} &= \sum_{\substack{0\le t_1,\cdots,t_r\le k-i\\ t_1+\cdots+t_r=k-i}} h_{t_1}\cdots h_{t_r}\\ &= \sum_{\substack{0\le t_1\le\cdots\le t_r\le k-i\\ t_1+\cdots+t_r=k-i}} \binom{r}{\beta_1,\cdots,\beta_\delta} h_{t_1}\cdots h_{t_r}.\end{aligned} \tag{3.10}$$

where distinct $q_1,\cdots,q_\delta$ satisfy $q_1\cdot\beta_1+\cdots+q_\delta\cdot\beta_\delta = t_1+\cdots+t_r = k-i$.

We consider when $j_{r+1} = k, j_1 = \cdots = j_r = 0$, thus have $0 \cdot r + k \cdot 1 = k$ with $\sigma = 2, \alpha_1 = r$ and $\alpha_2 = 1$, Equation (3.9) yields:

$$
\begin{aligned}
[x^k](h^{r+1}) &= \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}\le k\\ j_1+\cdots+j_{r+1}=k}} \binom{r+1}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}} \\
&= \frac{(r+1)!}{r!1!} h_0\cdots h_0 h_k + \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}< k\\ j_1+\cdots+j_{r+1}=k}} \binom{r+1}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}} \\
&= (r+1) f_0 h_k + \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}< k\\ j_1+\cdots+j_{r+1}=k}} \binom{r+1}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}} \\
&= (r+1) f_0 h_k + \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}< k\\ j_1+\cdots+j_{r+1}=k}} \frac{r+1}{k}\cdot k\binom{r}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}} \\
&= (r+1) f_0 h_k + M.
\end{aligned}
$$

We do some modification in the running index of M:

$$
\begin{aligned}
M =& \frac{r+1}{k} \sum_{\substack{0\le j_1\le\cdots\le j_{r+1}< k\\ j_1+\cdots+j_{r+1}=k}} k\cdot\binom{r}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_{r+1}} \\
=& \frac{r+1}{k} \sum_{j_{r+1}=0}^{k-1} \Bigg(\sum_{\substack{0\le j_1\le\cdots\le j_r\le k-j_{r+1}\\ j_1+\cdots+j_r=k-j_{r+1}}} k\cdot\binom{r}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_r}\Bigg) h_{j_{r+1}} \\
=& \frac{r+1}{k} \sum_{i=0}^{k-1} \Bigg(\sum_{\substack{0\le j_1\le\cdots\le j_r\le k-i\\ j_1+\cdots+j_r=k-i}} k\cdot\binom{r}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_r}\Bigg) h_i \\
=& \frac{r+1}{k} \sum_{i=0}^{k-1} \Bigg(k\cdot \sum_{\substack{0\le j_1\le\cdots\le j_r\le k-i\\ j_1+\cdots+j_r=k-i}} \binom{r}{\alpha_1,\cdots,\alpha_\sigma} h_{j_1}\cdots h_{j_r}\Bigg) h_i \\
=& \frac{r+1}{k} \sum_{i=0}^{k-1} \Big(k\cdot f_{k-i}\Big) h_i \quad \text{by Equation (3.10).}
\end{aligned}
$$

Therefore we have:

$$\begin{aligned} r f_o h_k &= (r+1) f_0 h_k - f_0 h_k \\ &= ([x^k](h^{r+1}) - M) - f_0 h_k \text{ by Equation (3.6)} \\ &= ([x^k](h^{r+1}) - M) - ([x^k](h^{r+1}) - \textstyle\sum_{i=0}^{k-1} f_{k-i} h_i) \text{ by Equation (3.5)} \\ &= \textstyle\sum_{i=0}^{k-1} f_{k-i} h_i - M \\ &= \textstyle\sum_{i=0}^{k-1} f_{k-i} h_i - \frac{r+1}{k} \sum_{i=0}^{k} \Big(k \cdot f_{k-i}\Big) h_i \text{ as desired.} \end{aligned}$$

3.2 Algorithms for Sparse Perfect Powers

We have algorithms for the perfect power problem in the dense setting. However, the total cost can be exponential, regardless of sparsity. In this section we will present fast sparsity sensitive algorithms for the sparse case modified from Algorithm 1. We will discuss the $f_0 \neq 0$ and $f_0 = 0$ cases separately.

3.2.1 When $f_0 \neq 0$

Now we want to solve Problem 2.2.2 sparse polynomial perfect power problem by modifying Algorithm 1. Instead of computing $h_1, \cdots, h_s$ as in Algorithm 1, we need to figure out the next highest coefficient of h in each iteration. Suppose we have computed $h_0, \ldots, h_{k-1}$ for some $k > 0$. Let

$$\begin{aligned} \check{h} &= \sum_{i=0}^{k-1} h_i x^i, \\ \hat{h} &= \sum_{i=k+1}^{s} h_i x^{i-(k+1)}, \end{aligned}$$

so $h = \check{h} + h_k x^k + \hat{h} x^{k+1}$. We know that $f'\check{h} - rf\check{h}' \equiv 0 \bmod x^{k-1}$ since no coefficient of h of degree k or higher affects this equation. Then Equation (2.1) can be written as:

$$\begin{aligned}
0 &= f'h - rfh' \\
&= f'(\check{h} + h_k x^k + \hat{h} x^{k+1}) - rf(\check{h} + h_k x^k + \hat{h} x^{k+1})' \\
&= f'(\check{h} + h_k x^k + \hat{h} x^{k+1}) - rf(\check{h}' + k h_k x^{k-1} + \hat{h}' x^{k+1} + (k+1)\hat{h} x^k) \\
&\equiv f'\check{h} - rf\check{h}' - rk f h_k x^{k-1} \bmod x^k \\
&\equiv f'\check{h} - rf\check{h}' - rk f_0 h_k x^{k-1} \bmod x^k \\
&\equiv R x^{k-1} \bmod x^k,
\end{aligned}$$

for some $R \in \mathsf{F}$; Rx^{k-1} is the "**residual**" or error at this step. This is a "Newton-like" method. When $f_0 \neq 0$, we can then solve

$$h_k = R/(r f_0 k). \tag{3.11}$$

Equation (3.11) leads to the following algorithm:

Algorithm 3: SparsePerfectPowerNonzeroConstAttempt

Input: sparse polynomial $f \in \mathsf{F}[x]$ of degree n s.t. $f_0 \neq 0$ and integer $s, r \in \mathbb{Z}$ s.t. $n = rs$

Output: sparse polynomial $h \in \mathsf{F}[x]$ such that $f = h^r$

1 $h_0 = f_0^{1/r}$;
2 $h = h_0$;
3 $k = 1$;
4 **while** degree$(h) < s$ **do**
5 $\quad res = f'h - rfh'$;
6 $\quad k = \text{ldegree}(res) + 1$;
7 $\quad h_k = \frac{1}{r f_0 k} \cdot \text{coeff}(res, k-1)$;
8 $\quad h \mathrel{+}= h_k \cdot x^k$;
9 **end**
10 **return** h;

Note that the condition degree$(h) < s$ for the `while` loop in line 5 can also be ldegree$(res) < s + 1$.

Theorem 3.2.1. *Algorithm 3 takes $O(\tau(f)\tau(h)^2)$ operations.*

Proof. At every iteration the cost is dominated by the multiplication of sparse polynomials $f'\check{h}$ and $f\check{h}'$ in line 6. Using the "traditional" method, this costs $O(\tau(f)\tau(h))$ operations for each. Each iteration we recover one non-zero coefficient of h, so there are $\tau(h)$ iterations. Thus, the whole algorithm can be done with $O(\tau(f)\tau(h)^2)$ operations. □

We can we make the algorithm faster by modifying the computation of `res`. We can compare how `res` changes in each iteration as only `h` is updated each time. We use `hlo` to note `h` in the current iteration, so we have:

$$\begin{aligned} res &= f' \cdot (hlo + h_k \cdot x^k) - r \cdot f \cdot (hlo + h_k \cdot x^k)' \\ &= f' \cdot hlo + f' \cdot h_k \cdot x^k - r \cdot f \cdot hlo' - r \cdot f \cdot (h_k \cdot x^k)' \\ &= res' + f' h_k x^k - rf \cdot k h_k x^{k-1}, \end{aligned}$$

where `res'` is the residual computed in the previous iteration. And also note that initialization of `res` gives:

$$res = f' \cdot h_0 - rf \cdot h_0' = f' \cdot h_0.$$

Therefore we have the following modified algorithm:

Algorithm 4: SparsePerfectPowerNonzeroConst

Input: sparse polynomial $f \in \mathsf{F}[x]$ of degree n s.t. $f_0 \neq 0$ and integer $s, r \in \mathbb{Z}$ s.t. $n = rs$

Output: sparse polynomial $h \in \mathsf{F}[x]$ such that $f = h^r$

1 $h_0 = f_0^{1/r}$;
2 $h = h_0$;
3 $res = f' h_0$;
4 $k = 1$;
5 **while** ldegree$(res) < s + 1$ **do**
6 $\quad k = \text{ldegree}(res) + 1$;
7 $\quad h_k = \text{coeff}(res, k-1)/(r f_0 k)$;
8 $\quad h \mathrel{+}= h_k \cdot x^k$;
9 $\quad res \mathrel{+}= f' \cdot h_k x^k - rkf \cdot h_k x^{k-1}$;
10 **end**
11 **return** h;

Theorem 3.2.2. *Algorithm 4 takes $O(\tau(f)\tau(h))$ operations.*

Proof. In each iteration, the cost of computation of `res` in line 10 is reduced to $\tau(f)$ since $h_k x^k$ and $k h_k x^{k-1}$ only have one term and multiplying them to f takes $\tau(f)$. Also the condition for the `while` loop in line 6 is equivalent to degree$(h) < s$. Hence there are $\tau(h)$ iterations, so in total, there will be $O(\tau(f)\tau(h))$ operations. □

Hence Algorithm 4 is a sparsity-sensitive algorithm to solve the perfect power problem when $f_0 \neq 0$.

3.2.2 When $f_0 = 0$

When we consider the case that $f_0 = 0$, if we try to modify Algorithm 2 as what we did in the previous $f_0 \neq 0$ case, things may not work as Algorithm 2 recovers the coefficients of h from the highest degree to lower degrees, with which the modulo x^k trick will not work.

One possible method is to find the lowest exponents of f, say d, then we have $f = \overline{f} x^d$, for some $\overline{f} \in \mathsf{F}[x]$ with $\overline{f}(0) \neq 0$, if and only if $h = \overline{h} x^{d/r}$ where $\overline{h} \in \mathsf{F}[x]$ with $\overline{h}(0) \neq 0$. Then it goes back to the $f_0 \neq 0$ case. This suggests the following algorithm:

Algorithm 5: SparsePerfectPowerZeroConst

Input: sparse polynomial $f \in \mathsf{F}[x]$ of degree n s.t. $f_0 = 0$ and integer $s, r \in \mathbb{Z}$ s.t. $n = rs$

Output: sparse polynomial $h \in \mathsf{F}[x]$ such that $f = h^r$

1 $d = \text{ldegree}(f)$;
2 $\overline{f} = f/x^d$;
3 $\overline{h} =$ Algorithm 4_SparsePerfectPowerNonzeroConst$(\overline{f}, r, s)$;
4 $h = \overline{h} \cdot x^{d/r}$;
5 **return** h;

Theorem 3.2.3. *Algorithm 5 takes $O(\tau(f)\tau(h))$ operations.*

Proof. Line 2 takes $O(\tau(f))$ operations and line 4 takes $O(\tau(h))$ operations, so the total running time is dominated by line 3, which is $O(\tau(f)\tau(h))$. □

Hence Algorithm 5 is a sparsity-sensitive algorithm to solve the perfect power problem when $f_0 = 0$. We now claim we have solved Problem 2.2.2 sparse perfect power problem.

It is a direct interesting follow up question to ask whether this is the fastest possible algorithm. This suggests the following open problem:

Open Problem 3.2.4. *Does there exist an algorithm for sparse perfect power problem that takes $O(\tau(f)+\tau(h))$ operations? If so, this algorithm would be optimal, since the input has size $O(\tau(f)+\tau(h)$ and any algorithm must access its entire input.*

Algorithm 4 and Algorithm 5 actually compute the r-th root h of f assuming that $f = h^r$. When f does not equal to h^r for any h, the algorithm still produces an h_{comp}. Since we are using "residual" to recover h in each iteration, h_{comp} is actually a solution to $(f' - rf\mathcal{D})h \equiv 0 \bmod x^s$ while $(f' - rf\mathcal{D})h \neq 0$.

This suggests the following algorithm for the more general problem $f = h^r \bmod x^s$:

Algorithm 6: SparsePerfectPowerMod

Input: sparse polynomial $f \in \mathsf{F}[x]$ of degree n and integer $s, r \in \mathbb{Z}$
Output: sparse polynomial $h \in \mathsf{F}[x]$ such that $f = h^r \bmod x^s$

```
if f_0 ≠ 0 then
    h = Algorithm 4_SparsePerfectPowerNonzeroConst(f, r, s);
else
    h = Algorithm 5_SparsePerfectPowerZeroConst(f, r, s);
end
return h;
```

The total complexity is the same as the previous algorithms, i.e. $O(\tau(f)\tau(h))$.

More details about solving more general differential equations are discussed in the next chapter. This sparse modular algorithm will also be key in what follows to solve the polynomial decomposition for sparse polynomials.

We also need to consider how to check $f = h^r$ to verify the algorithm. First note that computing the $r-$th power of a polynomial is expensive, so we consider to check whether $f'h = rh'f$ and $\mathrm{lc}(f) = \mathrm{lc}(h)^r$. This idea has been discussed in Giesbrecht and Roche (2008). Note the total operation should be the complexity of sparse polynomial multiplication, which is $O(\tau(f)\tau(h))$.

3.3 Perfect Power Problem for the "Wild" Case

In the previous sections, we have developed algorithms to solve the perfect power problem based on Equation (3.2) and Equation (3.3). We note that both of the equations assume $r \neq 0$, which follows by the "tame" case assumption that $\mathrm{char}(F)$ does not divide r. In this section, we will deal with the perfect power problem for the "wild" case, i.e. when

$\mathrm{char}(F) = p$ divides r. Say $r = p^\alpha \cdot q$, for some integer α and $\gcd(p, q) = 1$. Recalling some properties for $a, b, c \in F$ and any $\alpha \geq 1$, we have: $(a + b)^{p^\alpha} = a^{p^\alpha} + b^{p^\alpha}$, and $(a + b + c)^p = a^p + b^p + c^p$. Then we can have:

$$\begin{aligned}
f &= h^r \\
&= (h_0 + h_1 x + \cdots + h_s x^s)^{p^\alpha \cdot q} \\
&= \left((h_0 + h_1 x + \cdots + h_s x^s)^{p^\alpha}\right)^q \\
&= \left(h_0^{p^\alpha} + (h_1 x)^{p^\alpha} + \cdots + (h_s x^s)^{p^\alpha}\right)^q \\
&= \left(h_0^{p^\alpha} + h_1^{p^\alpha} x^{1 \cdot p^\alpha} + \cdots + h_s^{p^\alpha} x^{s \cdot p^\alpha}\right)^q \\
&= \left(\tilde{h_0} + \tilde{h_1} x^{1 \cdot p^\alpha} + \cdots + \tilde{h_s} x^{s \cdot p^\alpha}\right)^q \\
&= \tilde{h}^q,
\end{aligned}$$

where $\tilde{h_i} = h_i^{p^\alpha}$. Note that p does not divide q, so we can solve $f = \tilde{h}^q$ using the algorithms developed before and then recover h using $\tilde{h}$. So this will lead to the following algorithm

for perfect power problem for the "wild" case:

Algorithm 7: WildPerfectPower

Input: polynomial $f \in \mathsf{F}[x]$, integer $s, r \in \mathbb{Z}$ s.t. $n = rs$, characteristic p of F, integers α, q such that $r = p^\alpha \cdot q$

Output: polynomial $h \in \mathsf{F}[x], h_2 \neq 0$ such that $f = h^r$

```
if in the dense settings then
    if f_0 ≠ 0 then
        h̃ = Algorithm 1_DensePerfectPowerNonzeroConst(f, q, sp^α);
    else
        h̃ = Algorithm 2_DensePerfectPowerZeroConst(f, q, sp^α);
    end
else
    in the sparse settings;
    if f_0 ≠ 0 then
        h̃ = Algorithm 4_SparsePerfectPowerNonzeroConst(f, q, sp^α);
    else
        h̃ = Algorithm 5_SparsePerfectPowerZeroConst(f, q, sp^α);
    end
end
Let I be the exponents of h̃;
h = 0;
for i ∈ I do
    h+ = h̃_i^{1/p^α} · x^{i/p^α};
end
return h;
```

This would solve the "wild" case, and the total cost is still dominated by the "tame" case algorithms, which is $O((sp^\alpha)^2)$ or $O(\tau(f)\tau(h))$ respectively for dense and sparse settings.

Previous algorithms for computing the r-th root of sparse polynomials requires $O((\tau(f)+r)^4 \log r \log n)$ operations in Giesbrecht and Roche (2008). Here we have solved the perfect power problem for sparse polynomials with a faster algorithm in $O(\tau(f)\tau(h))$ operations.

We also want to check $f = h^r$ in the "wild" case. Similarly as before, we can check $f'\tilde{h} = q\tilde{h}'f$ and $\mathrm{lc}(f) = \mathrm{lc}(\tilde{h})^q$ as p does not divide q. The complexity would be the same as $O(\tau(f)\tau(h))$.

3.4 Perfect Power Problem for Rational Functions

A further exploration would be Problem 2.2.7 the perfect power problem for rational functions. A rational function f has the form $f = f_1/f_2$ for some $f_1, f_2 \in \mathsf{F}[x]$ and $f_2 \neq 0$. Suppose $n = rs$ for some $r \in \mathbb{N}$. The goal now is to determine if $f = h^r$ for some rational function h, where $h = h_1/h_2$ for some $h_1, h_2 \in \mathsf{F}[x]$ and $h_1, h_2 \neq 0$. Assume $f = h^r$, similarly as Equation (2.1), we will have the following observation:

$$\begin{aligned} f' &= (h^r)', \\ (f_1/f_2)' &= r \cdot (h_1/h_2)^{r-1} \cdot (h_1/h_2)' \\ &= r \cdot (h_1/h_2)^{r-1} \cdot \frac{h_1' h_2 - h_1 h_2'}{h_2^2} \\ &= r \cdot (h_1/h_2)^{r} \cdot \frac{h_2}{h_1} \cdot \frac{h_1' h_2 - h_1 h_2'}{h_2^2} \\ &= r \cdot (f_1/f_2) \cdot \frac{(h_1/h_2)'}{h_1/h_2}, \\ \frac{(f_1/f_2)'}{f_1/f_2} &= r \cdot \frac{(h_1/h_2)'}{h_1/h_2}, \end{aligned}$$

which still gives us:

$$f'h = rfh'.$$

We can utilize the algorithms for perfect power problem to solve the rational case. Since f is a rational function, so first we need to reduce it to the lowest terms, i.e. find $\mathrm{fgcd} = \gcd(\mathrm{f}_1, \mathrm{f}_2)$, where we will have

$$f = \frac{f_1}{f_2} = \frac{\tilde{f}_1 \cdot \mathrm{fgcd}}{\tilde{f}_2 \cdot \mathrm{fgcd}} = \frac{\tilde{h}_1^r \cdot \mathrm{fgcd}}{\tilde{h}_2^r \cdot \mathrm{fgcd}}.$$

where $\gcd(\tilde{h}_1, \tilde{h}_2) = 1$. Now we can apply the previous algorithm to $\tilde{f}_1$ and $\tilde{f}_2$ separately to get $\tilde{h}_1$ and $\tilde{h}_2$. Suppose $\gcd(f_1, f_2) = \mathrm{fgcd}$ is given. Then we can have the following

algorithm:

Algorithm 8: SparseRationalPerfectPower

Input: rational polynomial $f = f_1/f_2, f_1, f_2 \in \mathsf{F}[x]$, $f_2 \neq 0$ of degrees n_1, n_2 separately, integer $r \in \mathbb{Z}$, the greatest common divisor fgcd=gcd(f_1, f_2) of degree n_0

Output: rational polynomial $h = h_1/h_2, h_1, h_2 \in \mathsf{F}[x], h_2 \neq 0$ such that $f = h^r$

```
1  f̃_1 =quo(f_1, fgcd);
2  f̃_2 =quo(f_2, fgcd);
3  if f̃_1(0) ≠ 0 then
4  |  h̃_1 = Algorithm 4_SparsePerfectPowerNonzeroConst(f̃_1, r, (n_1−n_0)/r);
5  else
6  |  h̃_1 = Algorithm 5_SparsePerfectPowerZeroConst(f̃_1, r, (n_1−n_0)/r);
7  end
8  if f̃_2(0) ≠ 0 then
9  |  h̃_2 = Algorithm 4_SparsePerfectPowerNonzeroConst(f̃_2, r, (n_2−n_0)/r);
10 else
11 |  h̃_2 = Algorithm 5_SparsePerfectPowerZeroConst(f̃_2, r, (n_2−n_0)/r);
12 end
13 h = h̃_1/h̃_2;
14 return h;
```

Note the algorithm returns one solution while there are infinitely many solutions in terms of multiples.

It is tricky to consider the running time of this algorithm. In total, the algorithm takes $O(\tau(\tilde{f}_1)\tau(\tilde{h}_1)+\tau(\tilde{f}_2)\tau(\tilde{h}_2))$. However, $\tau(\tilde{f})$ can be large even if $\tau(f)$ is small. For example, $f = x^n - 1 = (x-1)(x^{n-1}+x^{n-2}+\cdots+1) = (x-1)\tilde{f}$, where f is sparse but $\tilde{f}$ can be really dense. This means it is still challenging to have an input-sparsity-sensitive algorithm for rational polynomials.

Also note that this algorithm assumes the greatest common divisors of f_1, f_2 are given. When it is not given, it becomes hard as the first step is to compute the gcd. Theorem 2.1.7 proves that determining the gcd of two sparse polynomials is **NP**-hard, which means computing the gcd is also hard. Hence the original Problem 2.2.7 is still an open challenge.

In addition, Theorem 2.1.7 stated that determining whether two sparse polynomials are not relatively prime is equivalent to determining a sparse polynomial is square-free over integer field. Similarly, we might ask the following open problem for rational polynomials:

Open Problem 3.4.1. *Given sparse polynomials $f_1, f_2 \in F[x]$, not necessarily relatively prime. Suppose we have a polynomial-time algorithm to determine whether $f_1/f_2 = (h_1/h_2)^2$ for some polynomials $h_1, h_2 \in F[x]$. Then given sparse polynomials $f_3, f_4 \in F[x]$, does there exist polynomial-time algorithm to determine whether $gcd(f_3, f_4) = 1$?*

3.5 Sparse Polynomial Decomposition in the "Tame" Case

Solving the sparse perfect power problem Problem 2.2.2 modulo x^s leads to an algorithm for decomposition of sparse polynomials in the "tame" case. This is the first polynomial-time algorithm for this problem that we know of.

We first make use of a reduction of von zur Gathen (1990), and show that it does not change the sparsity of our problem. We also note that solving the sparse perfect power problem Problem 2.2.2 will lead to a faster algorithm for sparse polynomial decomposition in the "tame" case. As discussed in von zur Gathen (1990), if $f = g \circ h$ and α, β are the leading coefficients of f and h, then consider the affine linear transformation :

$$\bar{f} = \frac{f}{\alpha} = \frac{g(h)}{\alpha} = \frac{g\big(\beta \cdot \frac{h-h(0)}{\beta} + h(0)\big)}{\alpha} = \Big(\frac{g\big(\beta x + h(0)\big)}{\alpha}\Big) \circ \frac{h - h(0)}{\beta} = \bar{g} \circ \bar{h},$$

where $\bar{f} = \frac{f}{\alpha}$, $\bar{g}(x) = \frac{1}{\alpha} \cdot g\big(\beta x + h(0)\big), \bar{h} = \frac{h-h(0)}{\beta}$. In this way $\bar{f}, \bar{g}, \bar{h}$ are monic and $\bar{h}(0) = 0$. So we can assume f, g, and h are monic and $h(0) = 0$. To recover h, we follow the method of von zur Gathen (1990) using the reversal of polynomial.

Definition 3.5.1. *For a polynomial $f = f_0 + \cdots + f_{n-1}x^{n-1} + x^n \in F[x]$, we define the reversal of f to be:*

$$\tilde{f} = 1 + f_{n-1}x + \cdots + f_0 x^n = x^n \cdot f(\frac{1}{x}) \in F[x].$$

As discussed in von zur Gathen (1990), to solve the function polynomial decomposition $f = g \circ h$ where g, h are monic and $h(0) = 0$, we can instead solve $\tilde{f} \equiv \tilde{h}^r \bmod x^s$, then $h = x^s \cdot \tilde{h}(\frac{1}{x})$ is a solution candidate to $f = h^r$. Note that h is monic, so $\tilde{h}(0) = 1 \neq 0$, we can apply Algorithm 6 to obtain $\tilde{h}$. Given f and a candidate h, we can use interpolation

to find g such that $f = g \circ h$. This will suggest the following algorithm:

Algorithm 9: SparsePolyDecomp

Input: sparse polynomial $f \in \mathsf{F}[x]$ of degree n,integer $s, r \in \mathbb{Z}$ s.t $n = rs$
Output: sparse polynomials $g, h \in \mathsf{F}[x]$ such that $f = g \circ h$

1 $\alpha = \mathrm{lc}(f),\ \beta = \alpha^{1/r},\ h_0 = f_0^{1/r}$;
2 $\bar{f} = f/\alpha$;
3 $\tilde{f} = \mathrm{reversal}(\bar{f})$;
4 $\tilde{h} =$ Algorithm 6_SparsePerfectPowerMod$(\tilde{f}, r, s)$;
5 $\bar{h} = x^s \cdot \tilde{h}(1/x)$;
6 $h = \beta \cdot \bar{h} + h_0$;
7 // Next we recover g;
8 Fix a set $\mathcal{S} \subseteq F$ with $2n$ elements
9 Let $a_1 = h(0)$;
10 For i from 2 to r do
11 Choose a random $a_i \in S$ repeatedly until $h(a_i) \notin \{h(a_1), \ldots, h(a_{i-1})\}$;
12 Interpolate $g \in \mathsf{F}[x]$ from points $\{(f(a_1), h(a_1)), \ldots, (f(a_r), h(a_r))\}$;
13 **return** g, h;

The cost of steps 1-6 of this algorithm is $O(\tau(f)\tau(h))$.

Each evaluation of h requires $O(\tau(h) \log s)$ operations in F using repeated squaring to compute powers, and we thus expect to require $O(r\tau(h) \log s)$ operations in F for step 11.

We claim that each iteration of step 11 takes an expected 2 evaluations of h at some random a_i in $\mathcal{S}$. To see this, observe that $h(a_i) \in \{h(a_1), \ldots, h(a_{i-1})\}$ if and only if a_i is a root of the polynomial

$$\Gamma_i(x) = \prod_{j=1}^{i-1} (h(x) - h(a_j)) .$$

Γ_i has degree $(i-1)s$, thus the chance of hitting a root of Γ in $\mathcal{S}$ is at most $(i-1)s/2n < rs/2n \leq 1/2$. Thus an expected two choices at each iteration of Step 11 will be required.

Once we have found the r distinct evaluations of h in step 11, we also need r evaluation of f, which requires $O(r\tau(f) \log n)$, the remaining steps are from dense polynomial interpolation, which can be done using fast polynomial interpolation with $O(r \log^2 r)$ operations in F (see von zur Gathen and Gerhard (2013), Corollary 10.12).

Theorem 3.5.2. *Algorithm 9 above works as stated and requires* $O(\tau(f)\tau(h) + r \log^2 r + r\tau(h) \log s + r\tau(f) \log n)$ *operations in* F

Given $f \in F[x]$, for monic f, such that $f = g \circ h$, this algorithm computes the unique monic g, h with $h(0) = 0$. But what if no such decomposition exists. In the perfect power case, we used the "trick" that $f'h = rh'f$ when $f = h^r$, but we know of no such certificate for general polynomial decomposition.

When F is sufficiently large, we can do a randomized test that our decomposition is correct. Let $u = f - g \circ h$, then $u \in F[x]$ clearly has degree at most n. Fix a subset $\mathcal{S} \subset F$ of size at least $2n$. Then for random $\alpha \in \mathcal{S}$, we can quickly compute $u(\alpha) = f(\alpha) - g(h(\alpha))$, since f and h are sparse. If $f = g \circ h$ then $u(\alpha) = f(\alpha) - g(h(\alpha))$ will always be zero. But if $u \neq 0$, then $u(\alpha) = 0$ only when α is a root of the polynomial u. A polynomial u of degree at most n can only be zero at n points, so

$$\text{Prob}\{\alpha \in \mathcal{S}, \quad u(\alpha) = 0\} < n/\#\mathcal{S} < 1/2.$$

Repeating this experiment k times and always getting $f(\alpha) - g(h(\alpha)) = 0$ gives us confidence that $f = g \circ h$ with probability at least $1 - 1/2^k$, so we can increase our confidence as much as desired.

Nonetheless, there remains an open problem to come up with a deterministic certificate that $f = g \circ h$ for sparse f.

Open Problem 3.5.3. *Given sparse polynomials $f, h \in F[x]$ of degree n, s respectively, and $g \in [x]$ of degree r, give a deterministic or Las Vegas (error-free probabilistic) algorithm to determine whether $f = g \circ h$ using poly$(\tau(f), \tau(h), \log n, \log r, s)$ operations.*

Finally, when the characteristic of the field F divides r, our algorithm does not work. While we were able to modify our sparse perfect power algorithm to accommodate this case, it is not clear how to do this for polynomial decomposition. This leads to the following open question:

Open Problem 3.5.4. *Does there exist a fast algorithm for sparse polynomial decomposition for the "wild" case?*

Chapter 4

Generalization to Differential Equations over Formal Power Series

In the previous chapter, we presented fast algorithms to compute the r-th root h of a given sparse polynomial f using (3.1) obtained by $f'h = rfh'$. We noted that solving Problem 2.2.2, the sparse perfect power problem, is in fact solving a differential equation over power series:

$$0 = (f' - rf\mathcal{D})h,$$

where $(f' - rf\mathcal{D})$ can be viewed as a differential operator. Consider the linear differential operator with polynomial coefficients defined as a polynomial

$$\mathcal{L} = f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2 + \cdots + f_\ell\mathcal{D}^\ell \in \mathsf{F}[x][\mathcal{D};'],$$

for $f_0, \ldots, f_\ell \in \mathsf{F}[x]$ and $f_\ell \neq 0$, $\ell \in \mathbb{Z}$ as the order of $\mathcal{L}$. In this chapter, we consider how to solve Problem 2.4.5, i.e. find the solution for

$$\mathcal{L}h \equiv 0 \bmod x^m.$$

We also assume $f_\ell(0) \neq 0$ in this chapter to prove theorems and develop algorithms. In Section 4.1, we prove the existence and the uniqueness of the solution using the undetermined coefficient method. In Section 4.2, we discuss how the perfect power problem serves as a special case for $\ell = 1$ to inspire sparse-sensitive algorithm to solve Problem 2.4.5 when $\ell = 1$ or 2. In Section 4.3, we present fast algorithm for general ℓ and discuss the open questions if $f_\ell(0) = 0$. Note that in this chapter, we use $f_i \in F[x]$ to note the polynomial coefficient of $\mathcal{L}$, and $f_{i,j} \in F$ to note the j-th coefficient of f_i.

4.1 Undetermined Coefficient Method

Before we jump to designing algorithms, we need to make sure the solution exists.

We consider the $\ell = 2$ case, which is discussed in Birkhoff and Rota (1991):

$$\mathcal{L}h = (f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2)h = 0. \tag{4.1}$$

Theorem 4.1.1 (Theorem 1 of Chapter 4 in Birkhoff and Rota (1991))**.**

Given h_0, h_1, assume $f_2(0) \neq 0$, Equation (4.1) has a unique power series solution.

Proof. Here we restate the proof in the manner to enlighten the follow up theorem for the generalized case. First note that since $f_2(0) \neq 0$, then there exists $f_2^{-1} \in F[x]$ such that $f_2^{-1}f_2 = 1$, so we can assume $f_2 = 1$ by multiplying f_2^{-1} to both sides of Equation (4.1), so now we need to solve:

$$f_0h + f_1h' + h'' = 0.$$

Similarly we can do term-by-term differentiation:

$$h'' = 2h_2 + 6h_3x + \cdots + (k+1)(k+2)h_{k+2}x^k + \cdots;$$

$$f_1h' = f_{1,0}h_1 + (2f_{1,0}h_2 + f_{1,1}h_1)x + \cdots + \Big(\sum_{i=0}^{k}(k+1-i)f_{1,i}h_{k+1-i}\Big)x^k + \cdots;$$

$$f_0h = f_{0,0}h_0 + (f_{0,0}h_1 + f_{0,1}h_0)x + \cdots + \Big(\sum_{i=0}^{k} f_{0,i}h_{k-i}\Big)x^k + \cdots.$$

Summing up these terms, we get:

$$\begin{aligned} h'' + f_1h' + f_0h = &\Big(2h_2 + f_{1,0}h_1 + f_{0,0}h_0\Big) + \\ &\Big(6h_3 + 2f_{1,0}h_2 + f_{1,1}h_1 + f_{0,0}h_1 + f_{0,1}h_0\Big)x + \cdots + \\ &\Big((k+1)(k+2)h_{k+2} + \sum_{i=0}^{k}(k+1-i)f_{1,i}h_{k+1-i} + \sum_{i=0}^{k} f_{0,i}h_{k-i}\Big)x^k + \cdots. \end{aligned}$$

Equate the coefficients of $1, x, \cdots, x^k, \cdots$ to zero, so we get:

$$h_{k+1} = -\frac{\sum_{i=0}^{k}(k-i)f_{1,i}h_{k-i} + \sum_{i=0}^{k} f_{0,i}h_{k-1-i}}{k(k+1)}, \forall k \geq 0.$$

□

This method can be generalized to:

$$\mathcal{L}h = (f_0 + f_1\mathcal{D} + \cdots + f_\ell \mathcal{D}^\ell)h = 0. \tag{4.2}$$

Theorem 4.1.2. *Given $h_0, \cdots, h_{\ell-1}$, assume $f_\ell(0) \neq 0$, Equation (4.2) has a unique power series solution.*

Proof. Similarly, if $f_\ell(0) \neq 0$, it is reduced to solve:

$$f_0 h + f_1 h' + \cdots + f_{\ell-1} h^{(\ell-1)} + h^{(\ell)} = 0.$$

With term-by-term differentiation:

$$\begin{aligned}
h^{(\ell)} &= \ell! h_\ell + \frac{(\ell+1)!}{1!} h_{\ell+1} x + \cdots + \frac{(\ell+k)!}{k!} h_{\ell+k} x^k; \\
f_{\ell-1} h^{(\ell-1)} &= (\ell-1)! f_{\ell-1,0} h_{\ell-1} + \Big(\frac{\ell!}{1!} f_{\ell-1,0} h_\ell + (\ell-1)! f_{\ell-1,1} h_{\ell-1} \Big) x + \cdots \\
&\quad + \Big(\sum_{i=0}^{k} \frac{(k+\ell-1-i)!}{(k-i)!} f_{\ell-1,i} h_{k+\ell-1-i} \Big) x^k + \cdots +; \\
&\vdots \\
f_1 h' &= f_{1,0} h_1 + (2 f_{1,0} h_2 + f_{1,1} h_1) x + \cdots + \Big(\sum_{i=0}^{k} (k+1-i) f_{1,i} h_{k+1-i} \Big) x^k + \cdots ; \\
f_0 h &= f_{0,0} h_0 + (f_{0,0} h_1 + f_{0,1} h_0) x + \cdots + \Big(\sum_{i=0}^{k} f_{0,i} h_{k-i} \Big) x^k + \cdots .
\end{aligned}$$

Add these equations together and equate the coefficients of $1, x, \cdots, x^k, \cdots$ to be zero, we will get:

$$\begin{aligned}
&- \frac{(\ell+k)!}{k!} h_{\ell+k} \\
&= \sum_{i=0}^{k} f_{0,i} h_{k-i} + \sum_{i=0}^{k} (k+1-i) f_{1,i} h_{k+1-i} + \cdots + \sum_{i=0}^{k} \frac{(k+\ell-1-i)!}{(k-i)!} f_{\ell-1,i} h_{k+\ell-1-i} \\
&= \sum_{i=0}^{k} f_{0,i} h_{k-i} + \sum_{t=1}^{\ell-1} \Big(\sum_{i=0}^{k} \big(\prod_{j=1}^{t} (k-i+j) \big) f_{t,i} h_{k+t-i} \Big).
\end{aligned}$$

Shift the index from $\ell + k$ to $1 + k$, therefore we will get:

$$h_{k+1} = -\frac{\sum_{i=0}^{k} f_{0,i} h_{k+1-\ell-i} + \sum_{t=1}^{\ell-1} \left(\sum_{i=0}^{k} \left(\prod_{j=1}^{t} (k+1-\ell-i+j) \right) f_{t,i} h_{k+1-\ell+t-i} \right)}{\frac{(k+1)!}{(k+1-\ell)!}}.$$

□

This means given $h_0, \cdots, h_{k-1}$, we can compute h_k. And this procedure continues as k goes to infinity. Our original Problem 2.4.5 considers modulo x^m, i.e. only compute finitely many h_k's, but the solution h is still uniquely determined. We have proved the existence and uniqueness of the solution h, we now consider algorithms to compute the solution.

4.2 The $\ell = 1, 2$ cases

We first consider the case that:

$$\mathcal{L} = f_0 + f_1 \mathcal{D}.$$

We want to find $h \in F[x]$ such that:

$$\mathcal{L}h = (f_0 + f_1 \mathcal{D})h = f_0 h + f_1 h' \equiv 0 \bmod x^m. \tag{4.3}$$

This is a simple generalization of the perfect power problem as we can take $f_0 = -f_1'/r$. So we still consider the previous"Newton-like" method to Equation (4.3), which gives:

$$\begin{aligned}
0 &= f_0 h + f_1 h' \\
&= f_0(\check{h} + h_k x^k + \hat{h} x^{k+1}) + f_1(\check{h} + h_k x^k + \hat{h} x^{k+1})' \\
&= f_0(\check{h} + h_k x^k + \hat{h} x^{k+1}) + f_1(\check{h}' + k h_k x^{k-1} + \hat{h}' x^{k+1} + (k+1)\hat{h} x^k) \\
&= f_0 \check{h} + f_1 \check{h}' + k f_1 h_k x^{k-1} \bmod x^k \\
&= f_0 \check{h} + f_1 \check{h}' + k f_1(0) h_k x^{k-1} \bmod x^k \\
&= R x^{k-1} \bmod x^k,
\end{aligned}$$

where $-Rx^{k-1}$ is the residual. When $f_1(0) \neq 0$, we will have:

$$h_k = -R/k f_1(0),$$

and similarly we can compute `res` as:

$$\begin{aligned} res &= f_0 \cdot (hlo + h_k \cdot x^k) + f_1 \cdot (hlo + h_k \cdot x^k)' \\ &= f_0 \cdot hlo + h_k \cdot f_0 \cdot x^k + f_1 \cdot hlo' + k \cdot h_k \cdot f_1 \cdot x^{k-1} \\ &= res' + h_k f_0 x^k + k h_k f_1 x^{k-1}. \end{aligned}$$

This suggest the following algorithm:

Algorithm 10: $\ell = 1$ Case

Input: polynomial $f_0, f_1 \in \mathsf{F}[x]$ of degree n_0 and n_1 respectively s.t. $f_1(0) \neq 0$ and integer $m \in \mathbb{Z}$

Output: polynomial $h \in \mathsf{F}[x]$ such that $f_0 h + f_1 h' \equiv 0 \bmod x^m$

```
1  Pick any h_0;
2  h = h_0;
3  res = f_0 h_0;
4  k = 1;
5  while ldegree(res) < m + 1 do
6  |   k = ldegree(res) + 1;
7  |   h_k = -coeff(res, k - 1)/(k f_1(0));
8  |   h += h_k · x^k;
9  |   res += f_0 · h_k x^k + k f_1 · h_k x^{k-1};
10 end
11 return h;
```

The correctness and cost is summarized as follows:

Theorem 4.2.1. *If $f_1(0) \neq 0$, then $\mathcal{L}h \equiv 0 \mod x^m$, where $\mathcal{L} = f_0 + f_1\mathcal{D}$, has a unique solution up to the constant multiple (this is also the reason that we can pick any h_0 as the starting point). The cost of finding h is $O(t \cdot \tau(h))$, where $\tau(f_i) \leq t, i = 0, 1$.*

Proof. For the running time, in each iteration, it is dominated by the computation of `res`, which takes $O(\max(\tau(f_0), \tau(f_1))$. The condition for the `while` loop in line 5 is equivalent to degree$(h) < m$. So there are $\tau(h)$ iterations, so in total, the cost will be $O(t \cdot \tau(h))$. □

Now we consider the higher order case $\ell = 2$:

$$\mathcal{L} = f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2.$$

Then we want to find $h \in F[x]$ such that:

$$\mathcal{L}h = (f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2)h = f_0h + f_1h' + f_2h'' \equiv 0 \bmod x^m. \tag{4.4}$$

Similarly we apply the "Newton-like" method to get:

$$\begin{aligned}
0 &= f_0h + f_1h' + f_2h'' \\
&= f_0(\check{h} + h_kx^k + \hat{h}x^{k+1}) + f_1(\check{h} + h_kx^k + \hat{h}x^{k+1})' + f_2(\check{h} + h_kx^k + \hat{h}x^{k+1})'' \\
&= f_0\check{h} + f_1\check{h}' + f_2\check{h}'' + kf_1h_kx^{k-1} + k(k-1)f_2h_kx^{k-2} + k(k+1)f_2\hat{h}x^{k-1} \mod x^{k-1} \\
&= \left(f_0\check{h} + f_1\check{h}' + f_2\check{h}''\right) + k(k-1)f_2(0)h_kx^{k-2} \mod x^{k-1} \\
&= Rx^{k-2} \mod x^{k-1},
\end{aligned}$$

where $-Rx^{k-2}$ is the residual. Also note that we are doing $\bmod x^{k-1}$, which is different from before. When $f_2(0) \neq 0$, we will have:

$$h_k = -R/k(k-1)f_2(0),$$

and similarly we can compute `res` as:

$$\begin{aligned}
res &= f_0 \cdot (hlo + h_k \cdot x^k) + f_1 \cdot (hlo + h_k \cdot x^k)' + f_2 \cdot (hlo + h_k \cdot x^k)'' \\
&= f_0 \cdot hlo + h_k \cdot f_0 \cdot x^k + f_1 \cdot hlo' + k \cdot h_k \cdot f_1 \cdot x^{k-1} + k(k-1) \cdot h_k \cdot f_2 \cdot x^{k-2} \\
&= res' + h_kf_0x^k + kh_kf_1x^{k-1} + k(k-1)h_kf_2x^{k-2}.
\end{aligned}$$

This approach is demonstrated in the following algorithm:

Algorithm 11: $\ell = 2$ Case

Input: polynomial $f_0, f_1, f_2 \in \mathsf{F}[x]$ of degree n_0, n_1 and n_2 respectively s.t. $f_2(0) \neq 0$ and integer $m \in \mathbb{Z}$

Output: polynomial $h \in \mathsf{F}[x]$ such that $f_0h + f_1h' + f_2h'' \equiv 0 \bmod x^m$

```
Pick any h_0, h_1;
h = h_0 + h_1 x;
res = f_0 h + f_1 h' + f_2 h'';
k = 1;
while ldegree(res) < m + 1 do
    k = ldegree(res) + 2;
    h_k = -coeff(res, k - 2)/(-k(k - 1) f_2(0));
    h += h_k · x^k;
    res += h_k f_0 x^k + k h_k f_1 x^{k-1} + k(k - 1) h_k f_2 x^{k-2};
end
return h;
```

The cost and correctness is summarized in the following theorem:

Theorem 4.2.2. *If $f_2(0) \neq 0$, then $\mathcal{L}h \equiv 0 \mod x^m$, where $\mathcal{L} = f_0 + f_1\mathcal{D} + f_2\mathcal{D}^2$, has a unique solution up to the constant multiple (this is the reason that we can pick any h_0 and h_1 as the starting point). The cost of finding h is $O(t \cdot \tau(h))$, where $\tau(f_i) \leq t, \forall i$.*

Proof. Similarly the running time is dominated by the computation of `res`, which takes $O(\max(\tau(f_0), \tau(f_1), \tau(f_2)))$. Also the condition for the `while` loop in line 5 is equivalent to degree$(h) < m + 1$, so in total the running time is still $O(t \cdot \tau(h))$. □

4.3 Polynomial differential equations of arbitrary order

Now we consider the general case:

$$\mathcal{L} = f_0 + f_1\mathcal{D} + \cdots + f_\ell\mathcal{D}^\ell.$$

We can modify the "Newton-like" method to get:

$$\begin{aligned}
0 &= f_0 h + f_1 h' + \cdots + f_\ell h^{(l)} \\
&= f_0(\check{h} + h_k x^k + \hat{h}x^{k+1}) + f_1(\check{h} + h_k x^k + \hat{h}x^{k+1})' + \cdots + f_\ell(\check{h} + h_k x^k + \hat{h}x^{k+1})^{(l)} \\
&= \left(f_0\check{h} + f_1\check{h}' + \cdots + f_\ell\check{h}^{(l)}\right) + \frac{k!}{(k-\ell)!}h_k f_\ell(0)x^{k-\ell} \mod x^{k-\ell+1} \\
&= Rx^{k-\ell} \mod x^{k-l+1},
\end{aligned}$$

where $-Rx^{k-\ell}$ is the residual. If $f_\ell(0) \neq 0$, we will have:

$$h_k = -R/\left(\frac{k!}{(k-\ell)!}f_\ell(0)\right).$$

Similarly we compute `res` as:

$$\begin{aligned}
res &= f_0 \cdot (hlo + h_k \cdot x^k) + f_1 \cdot (hlo + h_k \cdot x^k)' + \cdots + f_\ell \cdot (hlo + h_k \cdot x^k)^{(\ell)} \\
&= res' + h_k f_0 x^k + k h_k f_1 x^{k-1} + \cdots + \frac{k!}{(k-\ell)!}h_k f_\ell x^{k-\ell}.
\end{aligned}$$

This leads to the following algorithm:

Algorithm 12: GeneralCase

Input: polynomial $f_0, f_1, \cdots, f_\ell \in \mathsf{F}[x]$ of degree $n_0, n_1, \cdots, n_\ell$ respectively s.t. $f_\ell(0) \neq 0$ and integer $m \in \mathbb{Z}$

Output: polynomial $h \in \mathsf{F}[x]$ such that $f_0 h + f_1 h' + \cdots + f_\ell h^{(\ell)} \equiv 0 \bmod x^m$

1 Pick any $h_0, h_1, \cdots, h_{\ell-1}$;
2 $h = h_0 + h_1 x + \cdots + h_{\ell-1} x^{\ell-1}$;
3 $res = f_0 h + f_1 h' + \cdots + f_\ell h^{(\ell)}$;
4 $k = 1$;
5 **while** ldegree$(res) < m + 1$ **do**
6 $\quad k = \text{ldegree}(res) + \ell$;
7 $\quad h_k = -\text{coeff}(res, k - \ell)/(-\frac{k!}{(k-\ell)!} f_\ell(0))$;
8 $\quad h \mathrel{+}= h_k \cdot x^k$;
9 $\quad res \mathrel{+}= \sum_{i=0}^{\ell} \frac{k!}{(k-i)!} h_k \cdot f_i x^{k-i}$;
10 **end**
11 **return** h;

Theorem 4.3.1. *If $f_\ell(0) \neq 0$, then $\mathcal{L}h \equiv 0 \mod x^m$, where $\mathcal{L} = f_0 + f_1 \mathcal{D} + \cdots + f_\ell \mathcal{D}^\ell$, has a unique solution up to the constant multiple (this is the reason that we can pick any $h_0 \cdots h_{\ell-1}$ as the starting point). The cost of finding h is $O(t \cdot \ell \cdot \tau(h))$, where $\tau(f_i) \leq t, \forall i$.*

Proof. The running time is still dominated by the computation of `res`, which takes $O(t \cdot \ell)$. The condition of the `while` loop in line 5 is equivalent to degree$(h) < m + \ell - 1$, thus in total, it takes $O(t \cdot \ell \cdot \tau(h))$ operations. □

Similarly to Open Problem 3.2.4, it is interesting to think about the following open problem:

Open Problem 4.3.2. *Does there exist an algorithm to solve the generalized differential equation that takes $O(t + \ell + \tau(h))$ operations? If so, this algorithm should be optimal.*

Note in this chapter we assume $f_\ell(0) \neq 0$ to prove the existence and the uniqueness and develop the algorithms. It is interesting to think about the case when $f_\ell(0) = 0$. Our algorithm would still work for a sub-case when x^k divides all of $f_0, \cdots, f_\ell$ and $\frac{f_\ell}{x^k}(0) \neq 0$, this can be reduced to the nonzero case if k is given, and the total cost would still be $O(t \cdot \ell \cdot \tau(h))$ as the sparsity of each f_i does not change if divided by x^k.

A very challenging case is: if the leading polynomial coefficient still have zero constant after dividing $f_0, \cdots, f_\ell$ by x^k for the largest value of k. The algorithm will not work as we have $f_\ell(0)$ in the denominator. This address the following open question:

Open Problem 4.3.3. *Does there exist a sparsity sensitive algorithm to completely solve the $f_\ell(0) = 0$ case for finding polynomial solutions modulo x^m?*

Chapter 5

Implementations and Experiments

Part of the goal was to find example classes where sparse solutions to differential equations exist. One such example is Equation (2.1), where the sparsity of solutions is supported by the deep theory of Zannier and Schinzel (2009). Unfortunately, we ultimately did not identify any other interesting general classes of differential equations, but the search is ongoing, and this thesis provides the tools to solve them. This chapter will describe the implementation of Algorithm 12 to find the solution h to the general linear differential equation with sparse polynomial coefficients. Although the algorithm works for any finite field, the implementation is over $\mathbb{Z}$ computationally. The implementation and the tests are written in `Maple2020` and the source code is available in https://github.com/SaiyueLyu/sparsesoln.

5.1 Implementation and Tests

The `Maple` procedure will compute the solution h to the differential equation $\mathcal{L}h \equiv 0 \bmod x^m$, and will take three parameters. `flist` is the input list of sparse polynomials f_i's, `m` is the degree of the expected solution h, `hlo` is the starting point of h. The implementation of Algorithm 12 is called `sparse_de_solver` in the source codes.

To test Algorithm 12, we will need to pre-compute a set of sparse polynomials f_i's and $f_\ell(0) \neq 0$. Then we will run several trials of each setting and record the average sparsity of the solution in order to obtain a plot to visualize the trend of the sparsity of the computed solution h.

The first step is to consider generate sparse f_i's, here are 5 structures to build up some examples :

- Structure a : f_i's are random generated polynomials.
- Structure b : f_{i+1} is generated from f_i by exponents shifting, i.e. $f_0 = \sum_j c_{0,j} x^{e_j}$, $f_1 = \sum_j c_{1,j} x^{e_j+1}$, $f_2 = \sum_j c_{2,j} x^{e_j+2}$, $\cdots$ and $\tau(f_i) = \tau(f_{i+1})$ accordingly.
- Structure c : a special case of Structure b where $f_i = f'_{i+1}$.
- Structure d : the perfect power root special case, i.e. the order of the equation $\ell = 1$, where $f_0 = f'$ and $f_1 = -r \cdot f$ for some polynomial f and r is the input power of $f = h^r$.

Each structure is implemented such that $f_\ell(0) \neq 0$ and structure d is a special case for perfect power problem. To test the result for each structure, a `Maple` procedure `structure_i_flist` $(i = a, b, c)$ is implemented to generate a list of f_i's and takes three parameters :

- `l` as the order of the differential equation $\mathcal{L}h \equiv 0 \bmod m$,
- `tau` the sparsity bound of all f_i's,
- `d` the exponents bound of all f_i's.

Then a trail procedure `trail_i` will make multiple trails for the specified structure and return the average sparsity of the computed h, taking 5 parameters :

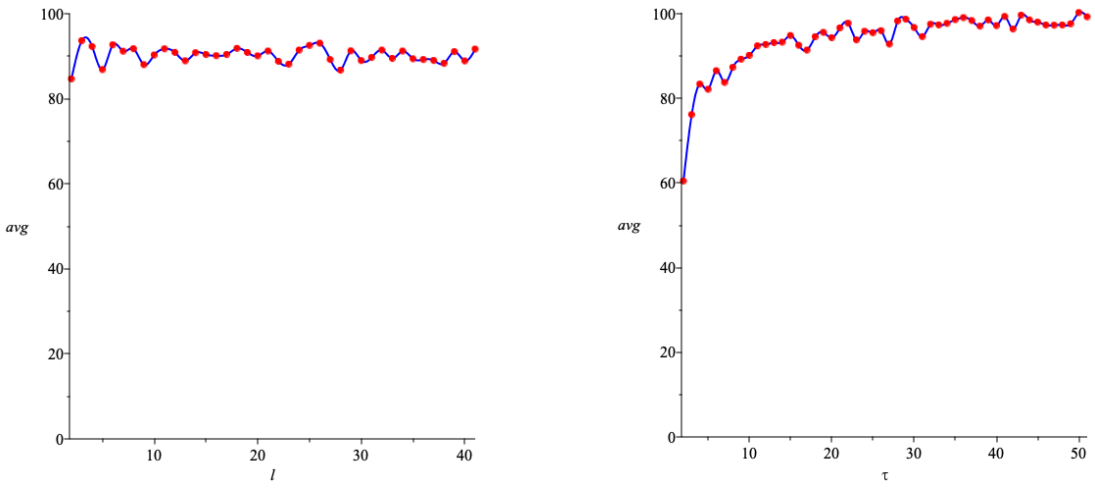
- `m` the degree of the expected solution h,
- `num` the total number of trails,
- `l` as the order of the differential equation, ℓ,
- `tau` the sparsity bound,
- `d` the exponents bound.

Also a procedure `plotl_i` outputs a list of average sparsity and a list of `l` values to see how the solution sparsity changes when the order `l` changes, taking 5 parameters : `m,num,tau,d,len`, where `len` is the length of the list of `l`. And a procedure `plottau_i` outputs a list of average sparsity and a list of `tau` values to see how the solution sparsity changes when the sparsity bound of f_i's changes, taking 5 parameters : `m,num,l,d,len`. Then we can utilize these procedures to output plots to visualize the relationship between the $\tau(h), \tau(f)$ and ℓ.

5.2 Experiment Results

We set `m`=100, `num`=50, `tau`=10, `d`=100 for `plotl_i`; and we set `m`=100, `num`=50, `l`=10, `d`=100 for `plottau_i`. Below illustrates the experiment results.

5.2.1 Structure a



(a) ℓ v.s avg sparsity of h

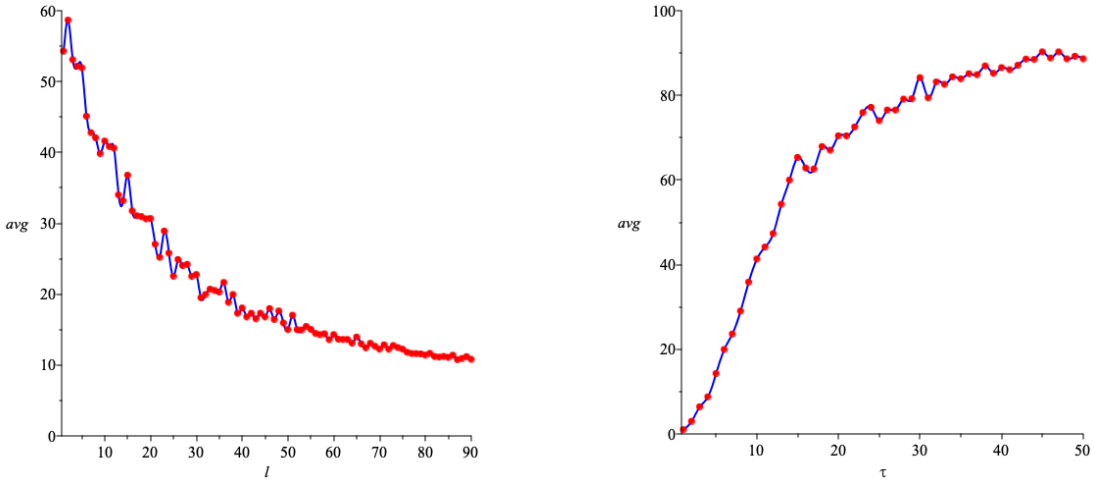
(b) τ v.s avg sparsity of h

Figure 5.1: Experiment results for structure a

Figure 5.1a shows the impact of the order of differential equation ℓ on the trend of average sparsity of h; Figure 5.1b shows the impact of the sparsity bound of f_i's τ on the trend of average sparsity of h.

We can observe from Figure 5.1a that the sparsity of h is always quite large (compared to the degree of h, $m = 100$) no matter how `l` changes. And according to Figure 5.1b, when `tau` increase, the sparsity of h increases sharply and then converges to around $100(= m)$. This suggests that for randomly generated sparse polynomials f_i's, the solution density is high.

5.2.2 Structure b and Structure c



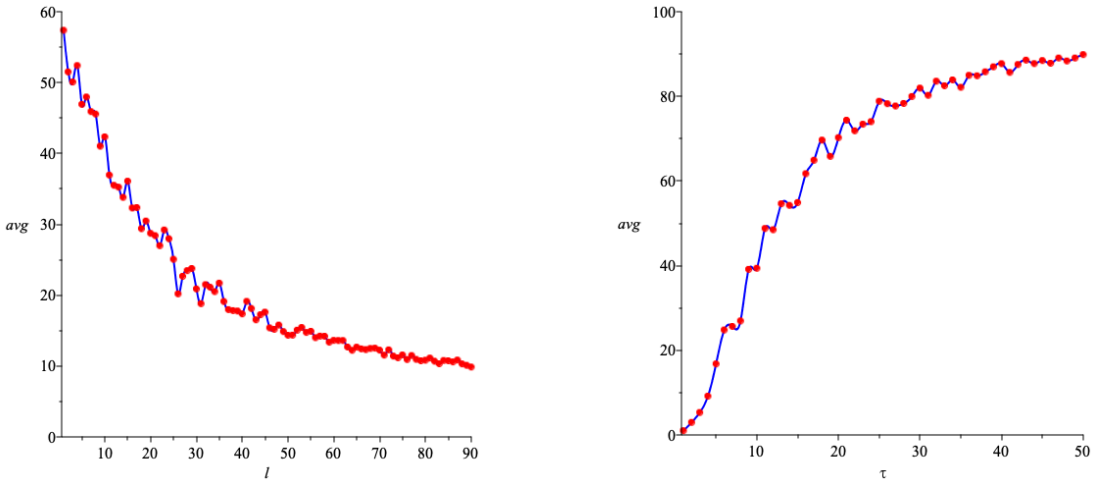
(a) ℓ v.s avg sparsity of h

(b) τ v.s avg sparsity of h

Figure 5.2: Experiment results for structure b

Figure 5.2 shows a quite different result compared to structure a. We can observe from Figure 5.2a that the starting point of the sparsity of h is relatively low compared to structure a, it decreases rapidly and then converges to around 10, which is around 10% of the degree m. And according to Figure 5.2b, the sparsity of h starts at 0, increases gradually and then converges but does not arrive 100. Figure 5.3 presents similar results as in Figure 5.2. These suggest the following observation, though formal proofs are not known :

Observation 5.2.1. *When the order of the differential equation increases, the solution sparsity decreases; When the sparsity of f_i's increases, the solution sparsity increases. And the coefficients of f_i's do not change the results between structure b and c.*



(a) ℓ v.s avg sparsity of h (b) τ v.s avg sparsity of h

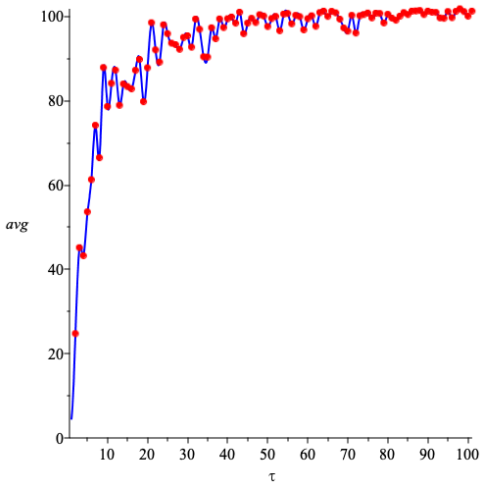
Figure 5.3: Experiment results for structure c

5.2.3 Structure d : Perfect Power Problem

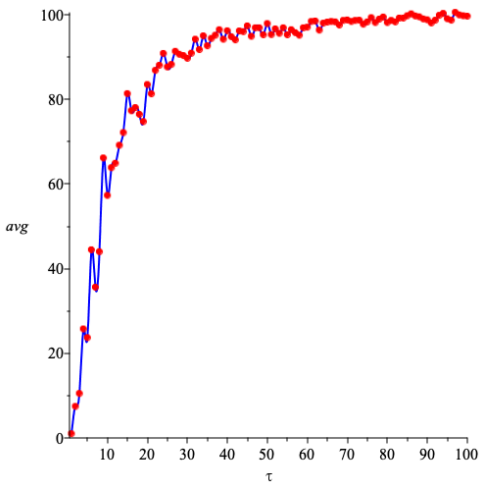
We then set up trails for perfect power problem, to compare structure d with structure a,b and c, we set `l`=1 and `r`=5. We want to see the relations regarding `tau`.

As illustrated in Figure 5.4, structure d makes the trend of solution sparsity look more linear than others, this suggests the following observation:

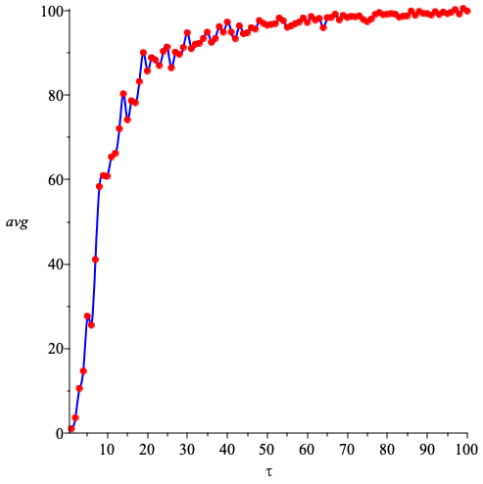
Observation 5.2.2. *The perfect power structure might lead the relationship between the sparsity of h and the sparsity of h^r to be more linear compared with other structures.*



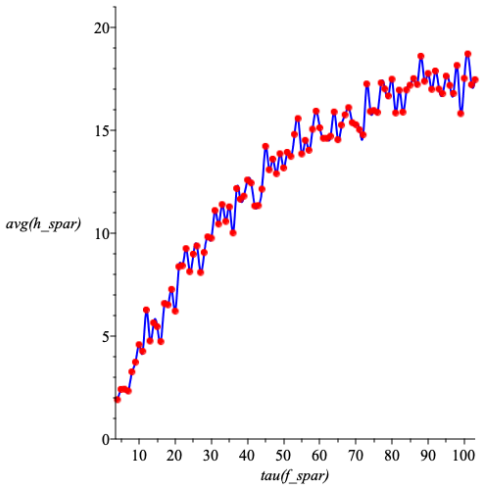
(a) τ v.s avg sparsity of h for structure a



(b) τ v.s avg sparsity of h for structure b



(c) τ v.s avg sparsity of h for structure c



(d) τ v.s avg sparsity of h for structure d

Figure 5.4: Perfect power experiment result comparison

Chapter 6

Conclusion and Future Work

In the previous chapters, we have explored Problem 2.2.2 as a starting point. Given a sparse polynomial over integers, rational numbers, or a finite field, we developed efficient algorithms to compute the r-th ($r \geq 2$) root of the given polynomial with polynomial time in terms of the sparsity of the input polynomial and the sparsity of the output polynomial. This technique was generalized to the first known sparsity-sensitive fast algorithm for polynomial decomposition of sparse polynomials. We provide a mathematical proof of correctness and complexity for all algorithms.

Our methods for sparse perfect powers and polynomial decomposition are based on a relationship with derivatives, suggested by Koiran (2011). This led to an exploration of a more general problem Problem 2.4.5. Given a linear differential equation with sparse polynomial coefficients, we developed efficient algorithm to solve $\mathcal{L}h \equiv 0 \bmod x^m$ with polynomial time in terms of the sparsity of the polynomial coefficients, the sparsity of the output and the order of the differential operator.

Several interesting problems remain unsolved. In Chapter 3, we have addressed the following open questions :

Open Problem 6.0.1. *Does there exist a fast algorithm for sparse polynomial decomposition for the "wild" case?*

A follow up problem would be to think of a series of polynomials for decomposition :

Open Problem 6.0.2. *Given monic $f \in F[x]$, determine whether there exists $g_1, \cdots, g_k \in F[x]$ such that $f = g_1 \circ \cdots \circ g_k$ such that each g_i is indecomposable.*

Open Problem 6.0.3. *Does there exist an algorithm for sparse perfect power problem that takes $O(\tau(f)+\tau(h))$ operations?*

Although we have shown we can utilize the algorithms for the perfect power problem to solve the rational case when the greatest common divisor of the input is also given, the total cost might not be input sparsity sensitive, and it remains futher exploration for the Problem 2.2.7 when the gcd is not pre-computed.

Open Problem 6.0.4. *Given rational function $f = f_1/f_2$ where f_1, f_2 are some sparse polynomials over finite field. If $gcd(f_1, f_2)$ is not given, is there an algorithm to find some rational $h = h_1/h_2$ such that $f = h^r$ with polynomial time in terms of $\tau(f_1), \tau(f_2), \tau(h_1), \tau(h_2)$?*

We also address a similarly open problem for rational case :

Open Problem 6.0.5. *Given sparse polynomials $f_1, f_2 \in F[x]$, not necessarily relatively prime. Suppose we have a polynomial-time algorithm to determine whether $f_1/f_2 = (h_1/h_2)^2$ for some polynomials $h_1, h_2 \in F[x]$. Then given sparse polynomials $f_3, f_4 \in F[x]$, does there exist polynomial-time algorithm to determine whether $gcd(f_3, f_4) = 1$?*

In Chapter 4, we solved the differential equation in $O(t \cdot \ell \cdot \tau(h))$ operations, we can ask the following open problem :

Open Problem 6.0.6. *Does there exist an algorithm to solve the generalized differential equation that takes $O(t + \ell + \tau(h))$ operations? If so, this algorithm should be optimal.*

In addition, the developed algorithms are based on the assumption that $f_\ell(0) \neq 0$. When $f_\ell(0) = 0$, we can not apply the modular trick to the newton-like algorithm, and the reversal trick by von zur Gathen (1990) might not be that helpful as in the perfect power case.

Open Problem 6.0.7. *When $f_\ell(0) = 0$, is there a fast algorithm to solve $\mathcal{L}h \equiv 0 \bmod x^m$?*

The experiment result in Chapter 5 suggest some observations of the relationship between the solution sparsity and the input sparsity or the order of the differential equation. Intuitively these observations make sense, but lack of rigorous proofs.

Open Problem 6.0.8. *Given the sparsity of the f_i's and the order ℓ of the differential equation $\mathcal{L}h \equiv \bmod x^m$, can we determine the sparsity of the solution h?*

More generally, do there exist other families of differential equations (beyond (2.1)) where we can expect sparse Taylor series, either provably or conjecturally.

References

Sergei A Abramov. m-sparse solutions of linear ordinary differential equations with polynomial coefficients. *Discrete Mathematics*, 217(1-3):3–15, 2000.

Sergei A Abramov and Anna A Ryabenko. Sparse power series and parameterized linear operators. *Programming and Computer Software*, 30(2):83–87, 2004.

David R Barton and Richard Zippel. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 1(2):159–168, 1985.

Garrett Birkhoff and Gian-Carlo Rota. *Ordinary Differential Equations*. Wiley, 1991.

Richard P Brent and Hsiang T Kung. Fast algorithms for manipulating formal power series. *Journal of the ACM (JACM)*, 25(4):581–595, 1978.

James Harold Davenport and Jacques Carette. The sparsity challenges. In *2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 3–7. IEEE, 2009.

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

Joachim von zur Gathen. Functional decomposition ofpolynomials: the tame case. *Journal of Symbolic Computation*, 9(3):281–299, 1990.

Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, third edition, 2013. ISBN 978-1-107-03903-2. doi: 10.1017/CBO9781139856065. URL http://dx.doi.org/10.1017/CBO9781139856065.

Mark Giesbrecht and Daniel S Roche. On lacunary polynomial perfect powers. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 103–110, 2008.

Mark Giesbrecht and Daniel S Roche. Detecting lacunary perfect powers and computing their roots. *Journal of Symbolic Computation*, 46(11):1242–1259, 2011.

Dima Yu Grigoriev, Marek Karpinski, and Andrew M Odlyzko. Existence of short proofs for nondivisibility of sparse polynomials under the extended riemann hypothesis. In *Papers from the international symposium on Symbolic and algebraic computation*, pages 117–122, 1992.

David Harvey and Joris Van Der Hoeven. Polynomial multiplication over finite fields in time o (n log n). 2019.

Marek Karpinski and Igor Shparlinski. On the computational hardness of testing square-freeness of sparse polynomials. In *International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, pages 492–497. Springer, 1999.

Pascal Koiran, 2011. Personal communication.

Dexter Kozen and Susan Landau. Polynomial decomposition algorithms. *Journal of Symbolic Computation*, 7(5):445–456, 1989.

Michael Monagan and Roman Pearce. Sparse polynomial division using a heap. *Journal of Symbolic Computation*, 46(7):807–822, 2011.

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

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

Joseph Fels Ritt. Prime and composite polynomials. *Transactions of the American Mathematical Society*, 23(1):51–66, 1922.

Daniel S Roche. What can (and can't) we do with sparse polynomials? In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation*, pages 25–30, 2018.

Andrzej Schinzel. On the number of terms of a power of a polynomial. *Acta Arithmetica*, 1(49):55–70, 1987.

David YY Yun. On square-free decomposition algorithms. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 26–35, 1976.

Umberto Zannier. On the number of terms of a composite polynomial. *Acta Arithmetica*, 127(2):157–167, 2007. URL http://eudml.org/doc/277948.

Umberto Zannier. On composite lacunary polynomials and the proof of a conjecture of Schinzel. *Inventiones Mathematicae*, 174:127–138, 2008.

Umberto Zannier and Andrzej Schinzel. On the number of terms of a power of a polynomial. *Rendiconti Lincei-Matematica e Applicazioni*, 20(1):95–98, 2009.