The Functional Decomposition of Polynomials

Matthew Thomas Dickerson
Ph.D Thesis

TR 89-1023
July 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501
THE FUNCTIONAL DECOMPOSITION OF
POLYNOMIALS

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Matthew Thomas Dickerson
August 1989
THE FUNCTIONAL DECOMPOSITION OF POLYNOMIALS

Matthew Thomas Dickerson, Ph.D.

Cornell University 1989

Let \( f(\bar{x}), h_1(\bar{x}), \ldots, h_d(\bar{x}) \) and \( g(\bar{x}) \) be elements of the polynomial ring \( K[\bar{x}] \) (or "polynomials over \( K \)""). If \( f(\bar{x}) = g(h_1(\bar{x}), \ldots, h_d(\bar{x})) \), then we call \( g, h_1, \ldots, h_d \) a functional decomposition of \( f \).

Polynomial decomposition is an important and interesting problem with a number of applications in computer science and computational algebra. Problems related to the decomposition of polynomials have received much attention in the past five years [AT85, BZ85, KL86, Dic87, vzGKL87, vzG87, vzG88, Dic88, KL89] as well as in the less recent past [Rit22, Eng41, Lev41, FM69, DW74]. In fact, the decomposition of polynomials is considered important enough that most major computational algebra systems (SCRATCHPAD II, MAPLE, MATHEMATICA) support polynomial decomposition for univariate polynomials.

In this thesis, we examine various problems related to the functional decomposition of polynomials. We will give a brief history of polynomial decomposition, and then give a number of new results, including the first polynomial time algorithms for two important decomposition problems and a proof of the NP-
completeness of another interesting polynomial decomposition problem. We will also discuss some of the applications of polynomial decomposition to problems such as polynomial factorization and computing the inverse of an automorphism over a multivariate polynomial ring. For the latter, we will give the first known polynomial time algorithm.
Biographical Sketch

Matthew Thomas Dickerson was born in Boston, Massachusetts, on June 12\textsuperscript{th}, 1963, the child of Willard Wilton Dickerson Jr. and Clara May Riddle Dickerson. After a brief and uneventful stay at the hospital, he returned to his parent’s home in nearby Cambridge where he lived until he reached the mature age of five. Then, deciding that he had had too much of city life, he took his family of five (his father and mother, and his two older brothers: Willard III and Melvin Theodore) to the slightly smaller town of Bryant Pond, Maine where he began his formal education in the Bryant Pond kindergarten. After a few months in Bryant Pond, he returned to Massachusetts to live in the town of Bolton where he spent the remainder of his all-too-brief youth.

During those “impressionable” years in Bolton, the author finally managed to gain a sister, Anh Thu, as well as two more brothers, Tuan and Thanh, all of whom came to join the Dickerson family from far-away Viet Nam. With five sons, the Dickersons were finally able to field their own men’s basketball team.

After graduating from Nashoba Regional High School in 1981, Mr. Dickerson went on to Dartmouth College in Hanover, New Hampshire, where he studied
Computer Science and Mathematics. He received his Bachelor of Arts degree from Dartmouth in June, 1985, graduating magna cum laude and with high honors from the department of Mathematics and Computer Science. From there, Mr. Dickerson proceeded to Cornell University in Ithaca, New York, to further his study in Computer Science. In 1986, he met the beautiful and wonderful Deborah Margaret Forrest, who on June 4, 1988 became his wife.
Dedicated to the glory of the living God and to His Son, the Messiah, Jesus of Nazareth, the Lamb of God to whom belongs the power, riches, wisdom, strength, honor, glory, and blessing now and forevermore.
Acknowledgements

Thanks to my parents, Willard and Clara May Dickerson, without whom this would not have been possible. And thanks to my wife, Deborah, without whom this would not have been enjoyable. And thanks to Deborah’s parents, Robert and Judy Forrest, without whom Deborah would not have been possible. And thanks to God who created all of the above with such care.

Thanks also to Dexter Kozen for lots of help and lots of ideas, many of which were good, and for turning me around on the numerous dead ends, and for patience. Also for the basketball games on Thursday mornings.

Finally, thanks to the faculty, staff, and students who make the Computer Science Department at Cornell as excellent as it is; particularly Gianfranco Bilardi for serving on my committee and Juris Hartmanis for being himself.

This work was supported, in part, by NSF grants DCR86-02663 and CCR88-06979.
## Table of Contents

1 Introduction
   1.1 Polynomial Decomposition: The Basic Problems .................. 4
   1.2 A History of Related Work ..................................... 7
   1.3 Applications of Polynomial Decomposition .................... 9
      1.3.1 Polynomial Factorization ................................. 9
      1.3.2 The N-Partition Problem ................................. 10
      1.3.3 Endomorphism Invertibility ............................ 12
   1.4 An Overview of Results ...................................... 16
   1.5 Notation .................................................. 17

2 Simple Multivariate Decomposition .............................. 19
   2.1 The Basic Algorithm ........................................ 20
      2.1.1 Background and Algorithm: the Case $d = 2$ .............. 20
      2.1.2 Analysis ............................................. 29
      2.1.3 Generalizing the Algorithm for Arbitrary $d$ ............ 31
      2.1.4 Decomposing Non-monic Polynomials ..................... 35
   2.2 A Speedup using Discrete Fourier Transforms ................ 35
      2.2.1 Background and Algorithm ............................. 36
      2.2.2 Analysis ............................................. 37
   2.3 Parallel Algorithms ........................................ 37
      2.3.1 NC Result ........................................... 39
      2.3.2 Practical Parallel Speedups ............................ 39
   2.4 Other Results .............................................. 41

3 Left Composition Factor of a Multivariate Decomposition .... 43
   3.1 Computing the Left Composition Factor ....................... 44
      3.1.1 Background and Algorithm: the Case $d = 2$ .............. 44
      3.1.2 An Explanation ........................................ 46
      3.1.3 Analysis ............................................. 49
      3.1.4 Generalizing the Algorithm to Arbitrary $d$ ............. 51
3.2 An Application: Automorphism Inversion in Polynomial Time . . 52
  3.2.1 The Inverse of an Automorphism . . . . . . . . . . . . . . . 53
  3.2.2 Background and Algorithm . . . . . . . . . . . . . . . . . . . . . 53
  3.2.3 Analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
  3.2.4 Endomorphism Invertibility: The Automorphism Question 57
3.3 Summary and some Other Results . . . . . . . . . . . . . . . . . . . . . 58

4 S-1-Decomposition 60
  4.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60
    4.1.1 Properties of the S-1-Decomposition . . . . . . . . . . . . . . . 60
    4.1.2 The 3-Partition problem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
  4.2 Proof of NP-hard . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
  4.3 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

5 Conclusions and Open Problems 68
  5.1 What Has Gone On Before: A Summary . . . . . . . . . . . . 68
  5.2 What is Left to be Done: A Look Ahead . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
    5.2.1 The Interesting, the Difficult, the Trivial, and the Nontrivial 70
    5.2.2 Open Problems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74

Bibliography 77
List of Figures

2.1 Algorithm 2.1 ........................................ 28
2.2 Algorithm 2.2 ........................................ 38
3.1 Algorithm 3.1 ........................................ 47
3.2 $C_k b_k = a_k$ ........................................ 48
3.3 Algorithm 3.2 ........................................ 55
Chapter 1

Introduction

Problems related to the functional decomposition of polynomials have received much attention in recent years [AT85,BZ85,KL86,Dic87,vzG87,vzGKL87,vzG88, Dic88,KL89], as well as in the less recent past [Rit22,Eng41,Lev41,FM69,DW74]. In this thesis, we will examine some polynomial decomposition problems and give both an overview of some results obtained elsewhere and a number of new results. Before discussing these problems and results, we define functional decomposition as it applies to polynomials.

Given a commutative ring $K$ with identity and indeterminates $x_1,\ldots,x_d$, the polynomial ring $K[x_1,\ldots,x_d]$ in $x_1,\ldots,x_d$ over $K$ is the well defined ring consisting of all polynomials in these $x_i$ with coefficients in $K$. Furthermore, if $f(x_1,\ldots,x_d) \in K[x_1,\ldots,x_d]$ is a polynomial, then $f$ defines a mapping of $\underbrace{K \times \cdots \times K}_d$ or $K^d$ into $K$. If $(a_1,\ldots,a_d) \in K^d$, then $f$ maps $(a_1,\ldots,a_d)$ into $f(a_1,\ldots,a_d)$. As in $K$, multiplication in $K[x_1,\ldots,x_d]$ is commutative.

Let $K$ be a commutative ring with identity and let $f(x), g(x), h(x) \in K[x]$
be univariate polynomials. If \( f(x) = g(h(x)) \), then we call \( g, h \) a functional decomposition of \( f \), and likewise we call \( f \) the functional composition of \( g, h \), also written \( f = g \circ h \). It is easy to see that for any unit \( a \in K \), the linear polynomials \( ax + b \) and \( x/a - b/a \) are inverses under composition. Thus for any \( a, b \in K \) with \( a \) a unit, we have a trivial decomposition of \( f(x) \) given by \( g(x) = ax + b \) and \( h(x) = f(x)/a - b/a \). A decomposition is called nontrivial if \( \text{deg} \, g, \text{deg} \, f > 1 \). A decomposition is called normal if \( h(0) = 0 \).

We can also define functional decomposition for multivariate polynomials. Let \( f(\vec{x}) \) and \( h(\vec{x}) \) be multivariate polynomials, and \( g(x) \) a univariate polynomial. If \( f(\vec{x}) = g(h(\vec{x})) \), then once again we call \( g, h \) a functional decomposition of \( f \). In this case as well we call the decomposition nontrivial if \( \text{deg} \, g > 1 \) and \( \text{deg}_{\text{tot}} h > 1 \) and we call the decomposition normal if \( h(0, \ldots, 0) = 0 \).

More generally, we can define functional decomposition with \( g \) also a multivariate polynomial. Let \( f(\vec{x}), h_1(\vec{x}), \ldots, h_n(\vec{x}) \in K[\vec{x}] \) and \( g(\vec{z}) \in K[\vec{z}] \) all be multivariate polynomials. If \( f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_n(\vec{x})) \), then we call \( g, h_1, \ldots, h_n \) a functional decomposition of a \( d \)-variate polynomial \( f \) into an \( n \)-variate polynomial \( g \) and \( n \) \( d \)-variate polynomials \( h_1, \ldots, h_n \). In this most general case, it is not obvious what the relationship between \( d \) and \( n \) should be, nor are the notions of trivial and nontrivial obvious. (We will save our discussion of which decompositions are trivial and nontrivial until a later chapter.)

Given polynomials \( g \) and \( h_1, \ldots, h_n \), it is clear that their functional composition \( f = g(\vec{h}) \) always exists, and the problem of computing that composition is quite trivial. It is much more interesting, as well as more difficult, to compute a nontrivial decomposition \( g, h_1, \ldots, h_n \) for a given polynomial \( f \). In fact, for a
random, univariate polynomial $f$, there is no guarantee that such a nontrivial, functional decomposition even exists. ($x^4 + 2x^3$, for example, has no nontrivial decomposition.) We therefore concern ourselves with the problem of determining when a decomposition exists, and when it does, of computing one.

The basic problems in polynomial decomposition usually take the following form:

**Problem 1 (General Polynomial Decomposition)** Given polynomial $f(\overline{x}) \in K[\overline{x}]$ and some subset of the following: polynomial $g(\overline{x}) \in K[\overline{x}]$, polynomials $h_1(\overline{x}), \ldots, h_n(\overline{x}) \in K[\overline{x}]$, and templates specifying the form of polynomials $g$ and $h_1, \ldots, h_n$, decide if there exists a functional decomposition $g, h_1, \ldots, h_n$ of $f$ such that $g$ and $h_1, \ldots, h_n$ are in the form specified by the template. If so, compute those coefficients of $g$ and the $h_i$'s which were not given.

This form seems general enough to encompass all of the interest decomposition problems that we have examined. For instance in the simple univariate case, which we shall define shortly, the dimension of $g$ is specified by the template, as are the degrees of $g$ and $h_1$. We are asked to compute the coefficients of $g$ and $h_1$.

In the remainder of this chapter, we will give a description of the problems examined in this thesis, followed by a brief history of some recent work in functional decomposition. In the following chapters, we will give a number of new results, including the first polynomial time algorithms for two important versions of this problem and a proof of the NP-hardness of another interesting version of the problem, which leads to a proof of the NP-hardness of Problem 1.

We will also briefly discuss some of the applications of polynomial decomposi-
tion. There are a number of applications in various areas of computer science and computational algebra. Two such applications are to polynomial factoring, or root finding, and to computing the inverse of an automorphism over a polynomial ring. We will briefly discuss the former and show how, in certain cases, polynomial decomposition can greatly speed up polynomial factorization. We will also devote a section to the latter, where we give the first polynomial time algorithm for computing the inverse of an automorphism over a multivariate polynomial ring. This algorithm is based on a decomposition algorithm also given in this thesis.

1.1 Polynomial Decomposition: The Basic Problems

We begin with a list of some of the decomposition problems that we will discuss in the following chapters. These problems are all specific instances of Problem 1. The first problem is the least general but has received the most attention in the past five years. We will briefly discuss some of the known results for this problem but will introduce no new results in this thesis.

Problem 2 (Univariate Decomposition) Given a monic polynomial $f(x) \in K[x]$ of degree $n$ and integers $r$ and $s$ such that $n = rs$ and $r, s > 1$, decide if there exists a functional decomposition $g, h$ of $f$ with $\deg g = r$ and $\deg h = s$. If so, determine the coefficients of $g$ and $h$.

It is easy to see that if any such decomposition $g, h$ exists for $f$, then there is also a decomposition $g', h'$ in which $g'$ and $h'$ are monic. To see this, let $g(x) = b_r x^r + \cdots + b_0$ and $h(x) = c_s x^s + \cdots + c_0$ with $c_s \neq 1$. We let $g'(x) = g(c_s x)$
and $h'(x) = h(x)/c_s$, giving us $g'(h'(x)) = g(c_s h(x)/c_s) = g(h(x)) = f(x)$. $h'$ is clearly monic, thus $g'$ is monic as well. Furthermore, $b_r c_s^r = 1$ so $c_s$ is invertible, thus if $h$ exists, then $h'(x) = h(x)/c_s$ exists as well. It can also be shown, as a result of Ritt's first theorem [Rit22], that if any decomposition exists, then there is a unique monic normal decomposition. In addition, it was also shown by Fried and MacRae [FM69] that if $K$ is a field and $f(x) \in K[x]$ is decomposable over an algebraic extension of $K$, then it is decomposable over $K$.

The following simple multivariate decomposition problem is the first generalization of the univariate decomposition problem and is the subject of Chapter 2.

**Problem 3 (Simple Multivariate Decomposition)** Given multivariate polynomial $f(\bar{x}) \in K[\bar{x}]$ of degree $\overline{n \times \cdots \times n}$ (that is: $\deg_{\text{tot}} f = nd$, or the term of highest degree in $f$ has degree $nd$, and $\deg_{x_i} f = n$ for $1 \leq i \leq d$, or $x_i$ is of degree $n$ in $f$) with $f$ monic (that is: the coefficient of $x_1^n \cdots x_d^n$ is 1) and given integers $r$ and $s$ such that $n = rs$ and $r > 1$, decide if there exists a functional decomposition $g, h$ of $f$ with $g$ univariate, $\deg g = r$, and $\deg h = s \times \cdots \times s$. If so, determine the coefficients of $g$ and $h$.

A more general decomposition problem allows the polynomial $g$ to be multivariate as well the polynomials $f, h_1, \ldots, h_d$. We may be given $f$ and asked to compute $g$ and the $h_i$'s; we may given both $f$ and $g$ and asked to compute the $h_i$'s; or, as in the following problem which we will discuss in Chapter 3, we may be given the polynomial $f$ as well as the polynomials $h_1, \ldots, h_d$ and asked to compute $g$.  

5
Problem 4 (Multivariate Left Composition) Given polynomials \( f(\vec{x}) \) and \( h_1(\vec{x}), \ldots, h_d(\vec{x}) \in K[\vec{x}], \vec{x} = x_1, \ldots, x_d \), and an integer \( r \), decide if there exists a polynomial \( g(\vec{x}) \in K[\vec{x}] \) of total degree at most \( r \) that composes with the \( h \)'s to give \( f \). That is, does there exist a polynomial \( g(\vec{x}) \) such that \( f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_d(\vec{x})) \) and \( \deg g \leq r \)? If so, determine the coefficients of \( g \).

This problem is really an instance of the subalgebra membership problem. We have an algebra (or \( K \)-algebra) \( K[\vec{x}] \) and a subset \( H = \{ h_1, \ldots, h_d \} \) of \( K[\vec{x}] \). \( K[h_1, \ldots, h_d] \) is the intersection of all subalgebras of \( K[\vec{x}] \) containing \( H \), also called the subalgebra generated by \( H \). (Equivalently, \( K[h_1, \ldots, h_d] \) is the smallest such subalgebra of \( K[\vec{x}] \) containing \( H \).) \( K[h_1, \ldots, h_d] \) is a subalgebra consisting of all elements \( f \) of the type:

\[
f = \sum a_{i_1, \ldots, i_d} h_1^{i_1} \cdots h_d^{i_d}
\]

with \( a_{i_1, \ldots, i_d} \in K \) and the sum ranging over a finite set of \( n \)-tuples \((i_1, \ldots, i_d)\) of integers \( \geq 0 \). Given a particular \( f' \in K[\vec{x}] \), we would like to know if \( f' \in K[h_1, \ldots, h_d] \), the subalgebra generated by the subset \( H \). If so, then there exists a polynomial \( g \) such that:

\[
f' = g(\vec{h}) = \sum a_{i_1, \ldots, i_d} h_1^{i_1} \cdots h_d^{i_d}.
\]

One can see that this is precisely what Problem 4 asks.

A final polynomial decomposition problem that we shall examine in this thesis is the \( s \)-1-decomposition problem. We define this decomposition problem below.

**Definition 1.1 (\( s \)-1-decomposition)** Let \( K \) be a field and let \( f(x) \) and \( h(x) \in K[x] \) be monic univariate polynomials of degree \( n = rs \) and \( s \), respectively. Let \( g(y, x) \in K[y, x] \) be a bivariate polynomial of the form:
\[ g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i) \quad (1.1) \]

with \( \alpha_i, \beta_i \in \bar{K} \), an algebraic extension of \( K \). If \( f(x) = g(h(x), x) \) then we call \( g, h \) an \( s \)-1-decomposition of \( f \), and likewise we call \( f \) an \( s \)-1-composition of \( g, h \). \( \square \)

In Chapter 4, we will examine the following \( s \)-1-decomposition problem and show that it is \( \text{NP} \)-hard.

**Problem 5 (S-1-decomposition Problem)** *Given univariate polynomial \( f(x) \) and an integer \( s \), decide if there exists an \( s \)-1-decomposition \( g, h \) of \( f \). If so, determine the coefficients of \( g \) and \( h \).*

### 1.2 A History of Related Work

Over the past five years, there have been a number of new results in the area of polynomial decomposition. We now give a brief history of some of these recent results.

In 1985, both Barton and Zippel [BZ85] and later Alagar and Thanh [AT85] gave algorithms for the solution of Problem 2 - obtaining, when one exists, a nontrivial functional decomposition \( g, h \) of a polynomial \( f(x) \in K[x] \). In both cases, the algorithms were exponential time, required polynomial factorization, and only worked over fields of characteristic 0. In 1986, Kozen and Landau [KL86] gave the first *polynomial* time algorithms for the solution of this problem. Their first algorithm required \( O(r^3 s^2) \) or \( O(n^3) \) time while their second algorithm, which made use of the Fourier transform, required only \( O(s^2 r \log r) \) or \( O(n^2) \) time. A later version of that paper [KL89] improved the result to an \( O(n^2) \) time algorithm which does not depend on the Fourier transform. Their
algorithms worked over any commutative ring $K$ in the “tame case” when the ring $K$ contains a multiplicative inverse of $r$, the degree of $g$. Their decomposition algorithms were also shown, using a result from Valiant, Skyum, Berkowitz, and Rackoff [VSBR81], to parallelize so as to run in $O(\log^{O(1)} n)$ depth with polynomially many processors. In the same paper, Kozen and Landau also gave a structure theorem for testing decomposability over arbitrary fields as well as some decomposition results for the “wild case” when $r$ divides the characteristic of $K$.

In 1987, Dickerson [Dic87] gave the first polynomial time algorithm for Problem 3, the more general functional decomposition of multivariate $f$ into a univariate $g$ and a multivariate $h$. That algorithm was based on a generalization of the technique used in [KL86] and also worked over any commutative ring $K$ in the “tame case”. Chapter 2 is devoted to this algorithm and some recent improvements.

Von zur Gathen [vzG87, vzG88] has since given faster algorithms for both the univariate and multivariate decompositions in the “tame case” as well as some further partial results for the “wild case” when the characteristic of $K$ divides the degree of $h$. The univariate decomposition algorithm of [vzG87] requires only $O(n \log^2 n \log \log n)$ time or $O(n \log^2 n)$ time when $K$ supports a Fourier transform.

There were no previously known polynomial time algorithms for the more general case of the multivariate decomposition problem, Problem 1, when the polynomial $g$ is multivariate of dimension $\geq 2$. In the final chapter, we will discuss some difficult, interesting, trivial, and nontrivial forms of this problem.
1.3 Applications of Polynomial Decomposition

In this section, we discuss three applications of polynomial decomposition. These applications are the problem of polynomial factorization, the $n$-partition problem, and the problem of characterizing the class of automorphisms of $K[\bar{x}]$ and computing their inverses.

1.3.1 Polynomial Factorization

Polynomial decomposition has a number of applications. One such application, as pointed out in [BZ85], is to the problem of polynomial factorization. Polynomial factorization, or root finding, is a difficult problem of great interest to many fields of science. The application of polynomial decomposition to polynomial factorization is given below.

In many situations, we would like to factor a given univariate polynomial $f(x) \in K[x]$. Assume that $f$ has a functional decomposition $g, h$. Let $g(x)$ factor in an extension of $K$ as:

$$g(x) = \prod_{i=1}^{r} (x - \beta_i)$$

(1.2)

and for $1 \leq i \leq r$ let $h(x) - \beta_i$ also factor in an extension of $K$ as:

$$h(x) - \beta_i = \prod_{j=1}^{s} (x - \alpha_{ij}).$$

(1.3)

Then we can see that:

$$f(x) = \prod_{i=1}^{r} \prod_{j=1}^{s} (x - \alpha_{ij}).$$

(1.4)

From Equation 1.4 we see that if $f$ has a decomposition $g, h$, then we can factor $f(x)$ over an algebraic closure of $K$ by first factoring $g(x)$ and then factoring
\[ h(x) - \beta_i \text{ for } 1 \leq i \leq r. \] Let \( \mathcal{F}(n) \) be the time required to factor a polynomial of degree \( n \), and let \( \mathcal{D}(n) \) be the time required to decompose a polynomial of degree \( n \). Then using this method we get:

\[ \mathcal{F}(n) = \mathcal{D}(n) + \mathcal{F}(r) + r\mathcal{F}(s). \] (1.5)

For example, if \( r = s = n^{1/2} \) then we have:

\[ \mathcal{F}(n) \leq \mathcal{D}(n) + n^{1/2}\mathcal{F}(n^{1/2}). \] (1.6)

which is a significant improvement if \( n < \mathcal{D}(n) \ll \mathcal{F}(n) \).

If we can perform this algorithm recursively—that is, if the polynomials \( h(x) - \beta_i \) can also be factored using the same technique of decomposition—then a straightforward analysis gives

\[ \mathcal{F}(n) = O(n \log n). \] (1.7)

Given a "random" polynomial \( f(x) \), it is highly unlikely that \( f \) will decompose, but in those instances in which it does, we have a factorization algorithm based on decomposition which is significantly faster than the best known previous factorization algorithms. Furthermore, although it is unlikely that a random polynomial will decompose, it requires only \( O(n \log n) \) time to test if \( f(x) \) is decomposable, so the cost of determining if we can use this method is relatively low compared with the cost of factorization using other methods.

1.3.2 The \( N \)-Partition Problem

There is a similar application of polynomial decomposition to the \( n \)-partition problem, which is known to be NP-hard for \( n \geq 3 \) [GJ79]. Given a set \( A = \{a_1, \ldots, a_m\} \) and a rational-valued function \( q : A \rightarrow Q \), we want to compute a
partition of $A$ into $m$ disjoint subsets $B_1, \ldots, B_m$ of $n$ elements each such that for $1 \leq i \leq m$ we have:

$$
\sum_{a_j \in B_i} q(a_j) = \frac{1}{m} \sum_{a_j \in A} q(a_j).
$$

(1.8)

In certain instances, we can use polynomial decomposition to solve the $n$-partition problem. Given an instance of the $n$-partition problem—a set $A = \{a_1, \ldots, a_{mn}\}$ and a rational-valued function $q$—define polynomial $f(x) \in Q[x]$ as follows:

$$
f(x) = \prod_{i=1}^{mn} (x - q(a_i)).
$$

(1.9)

Assume that $f$ has a functional decomposition $g, h$ with $\deg g = m$ and $\deg h = n$ and let $g$ factor in an extension of $Q$ as:

$$
g(x) = \prod_{i=1}^{m} (x - \beta_i).
$$

(1.10)

Then

$$
f(x) = g(h(x)) = \prod_{i=1}^{m} (h(x) - \beta_i).
$$

(1.11)

Now for $1 \leq i \leq m$ let $h(x) - \beta_i$ also factor in an extension of $Q$ as:

$$
h(x) - \beta_i = \prod_{j=1}^{n} (x - \gamma_{i,j}).
$$

(1.12)

Now we see from Equations 1.9 and 1.12 that:

$$
f(x) = \prod_{i=1}^{mn} (x - q(a_i)) = \prod_{i=1}^{m} \prod_{j=1}^{n} (x - \gamma_{i,j}).
$$

(1.13)

Thus the multisets $\{q(a_1), \ldots, q(a_{mn})\}$ and $\{\gamma_{i,j} : 1 \leq i \leq m, 1 \leq j \leq n\}$ are equal.

From Equation 1.12, however, we see that for $1 \leq i \leq m$, the coefficient of $x^{n-1}$ in $h(x) - \beta_i$ is given by $\sum_{j=1}^{n} \gamma_{i,j}$. This coefficient is the same for all $h(x) - \beta_i$.

For $1 \leq i, k \leq m$, we thus have:
\[
\sum_{j=1}^{n} \gamma_{i,j} = \sum_{j=1}^{n} \gamma_{k,j}.
\] (1.14)

It follows that the roots of the \( h(x) - \beta_i \) give us the values of an \( n \)-partitioning of \( A \). We can compute this partitioning by computing first the polynomial \( f \), then the decomposition \( g, h \) of \( f \), then factoring \( g \), and finally factoring \( h - \beta_i \).

(We note here that it has been shown by Lenstra, Lenstra and Lovász [LLL82] that polynomials in \( Q[x] \) can be factored over \( Q[x] \) in deterministic polynomial time; by Berlekamp [Ber67] that for a fixed finite field \( K \), polynomials in \( K[x] \) can be factored over \( F[x] \) also in deterministic polynomial time; and by Ben-Or, Feig, Kozen, and Tiwari [BOFKT88] that the problem of factoring polynomials in \( R[x] \) with all real roots is in NC.)

As with the case with the previous application to polynomial factorization, it is highly unlikely that the polynomial \( f \) will decompose. In those instances in which it does, however, we can compute an \( n \)-partition of \( A \) in polynomial time.

### 1.3.3 Endomorphism Invertibility

A third application of polynomial decomposition is to the problem of endomorphism invertibility. A \( K \)-algebra homomorphism is a linear mapping \( \sigma : R \to R' \) for \( K \)-algebras \( R \) and \( R' \) such that \( \sigma(xy) = \sigma(x)\sigma(y) \) for all \( x, y \in R \) and which maps the unit element of \( R \) to the unit element of \( R' \). An endomorphism is an homomorphism of \( R \) onto itself. An automorphism is an endomorphism which is one-to-one and onto. That is, an automorphism \( \sigma \) is an endomorphism such that there exists an inverse \( \psi \) of \( \sigma \) with \( \psi \sigma = \sigma \psi \) the identity map.

We denote by \( \text{End}_K K[\bar{x}] \) the set of endomorphisms on \( K[x] \) which fix elements of \( K \). Similarly, we let \( \text{Aut}_K K[\bar{x}] \) be the automorphisms on \( K[x] \) which fix
elements of $K$. An endomorphism (or automorphism) $\sigma$ mapping $K[\vec{x}] \to K[\vec{x}]$ is completely determined by its values on $x_1, \ldots, x_n$, since $K[\vec{x}]$ is free on generators of $x_1, \ldots, x_n$. We can therefore specify $\sigma \in \text{End}_K K[\vec{x}]$ (or $\text{Aut}_K K[\vec{x}]$) by a set of polynomials $h_1, \ldots, h_d \in K[\vec{x}]$ such that $\sigma : x_i \mapsto h_i(\vec{x})$.

Closely related to $\text{Aut}_K K[\vec{x}]$ are the general linear group $GL(n, K)$, the affine transformation group $A(n, K)$, the Jonquièrre automorphisms $J(n, K)$, and the subgroup of nilpotency $N(n, K)$, all of which are subgroups of $\text{Aut}_K K[\vec{x}]$. We define these in the following paragraphs.

The general linear group $GL(n, K)$ consists of those automorphisms $\sigma$ of the form:

$$\sigma(x_i) = \alpha_{i,1}x_1 + \alpha_{i,2}x_2 + \ldots + \alpha_{i,n}x_n \quad (1.15)$$

where $\alpha_{i,j} \in K$ for $1 \leq i, j \leq n$.

The affine transformation group $A(n, K)$ consists of those automorphisms $\pi$ of the form:

$$\pi(x_i) = \sigma(x_i) + c_i \quad (1.16)$$

where $\sigma \in GL(n, K)$ and $c_i \in K$ for $1 \leq i \leq n$.

The Jonquièrre automorphisms $J(n, K)$ consists of those automorphisms $\tau$ of the form:

$$\tau(x_i) = a_i x_i + f_i(x_{i+1}, \ldots, x_n) \quad (1.17)$$

where $a_i$ are units in $K$ and $f_i(x_{i+1}, \ldots, x_n) \in K[x_{i+1}, \ldots, x_n]$ for $1 \leq i \leq n$ ($f_n \in K$).

Finally, the subgroup of nilpotency $N(n, K)$ consists of those automorphisms $\psi$ of the form $\psi(x_i) = x_i + g_i(\vec{x})$ for $1 \leq i \leq n$ where $g_1, \ldots, g_n$ are nilpotent.
elements of $K[\bar{x}]$. (Remember a nilpotent element $\alpha \in A$ is one such that $\alpha^n = 0$ for some integer $n \geq 1$.)

The endomorphism invertibility problem for polynomials is as follows:

**Problem 6 (Endomorphism Invertibility)** Given $\sigma \in \text{End}_K K[\bar{x}]$, determine if $\sigma$ is invertible, and if it is, compute its inverse.

Restated, Problem 6 is to determine when an endomorphism is an automorphism, and when it is, to compute its inverse.

Returning to the question of subalgebra membership, we mentioned that $\sigma$ is completely determined by its values on $x_1, \ldots, x_d$. Let $\sigma(x_i) = h_i(\bar{x})$ for $1 \leq i \leq d$. If $\sigma$ is an automorphism, then $K[\bar{x}] = K[h_1, \ldots, h_d]$ and $H = \{h_1, \ldots, h_d\}$ is a set of generators for $K[\bar{x}]$ over $K$. That is, $K[\bar{x}] = K[h_1, \ldots, h_d]$ if and only if $x_1, \ldots, x_d$ are all in the $K$-subalgebra generated by $H$. The Endomorphism Invertibility Problem, Problem 6, is to determine if $H$ is a set of generators for $K[\bar{x}]$ over $K$, or equivalently to determine if $x_1, \ldots, x_d$ are all in the $K$-subalgebra generated by $H$.

A subproblem of Problem 6 is the following:

**Problem 7 (Inverse of an Automorphism)** Given $\sigma \in \text{Aut}_K K[\bar{x}]$, compute its inverse.

Problems 6 and 7 have received much attention dating back as far as the time of Hilbert and are closely related to Hilbert's 14th problem [Nag65].

**Problem 8 (Hilbert's 14th Problem)** Let $K$ be a field and $G$ a subgroup of the general linear group $GL(n, K)$. Then $G$ acts as a group of automorphisms of
$K[\bar{x}]$. Let $I_G$ be the ring of elements of $K[\bar{x}]$ invariant under $G$. Is $I_G$ finitely generated over $K$?

One famous conjecture related to Problem 6 is:

**Conjecture 1.1** $\text{Aut}_K K[\bar{x}]$ is generated by $GL(n, K)$, $J(n, K)$, and $N(n, K)$.

A second famous conjecture related to Problem 6 is the *Jacobian Conjecture*, which was first formulated by O.H. Keller in 1939 [Kel39].

**Conjecture 1.2 (Jacobian Conjecture)** If $K$ contains the field $Q$ of rational numbers and $\sigma$ is an endomorphism of $K[\bar{x}]$ whose Jacobian matrix is invertible, then $\sigma$ is an automorphism of $K[\bar{x}]$.

Many mathematicians have worked on the problems of determining when an endomorphism is invertible and computing its inverse [Abh74, Cam73, Kel39, NS83, SS89]. In 1972, Masayoshi Nagata [Nag72] devoted an entire book to problem of characterizing the automorphism group. In 1982, Bass, Connell, and Wright [BCW82] devote a chapter of a fairly substantial paper simply to an historical account of work on the Jacobian Conjecture, including many partial results as well as a discussion of some faulty proofs that appeared in the late 1950's and early 1960's. Both Conjectures 1.1 and 1.2 have been proven true under certain conditions but are still open in the general case when dimension $d$ of $\bar{x}$ is arbitrary and when the field $K$ has arbitrary characteristic. For good summaries of Conjectures 1.1 and 1.2 as well as related work on characterizing the automorphism group, see [Nag72] and [BCW82] respectively.

In addition to characterizing the group of automorphisms, is it of great interest to actually compute the inverse of a given $\sigma$ in the case when the endo-
morphism $\sigma$ is an automorphism. Many mathematicians have worked on this problem as well. There are a number of inversion formulas which give $\sigma^{-1}$ in terms of the inverse of the Jacobian matrix. Lagrange Inversion Formulas such as that of Abhyankar [Abh74,BCW82] give a nice mathematical model for the inverse, but it and other similar formulas such as that of Nousiainen and Sweedler [NS83] may not terminate. Shannon and Sweedler [SS89] recently gave an elegant algorithm to compute $\sigma^{-1}$ which uses Gröbner bases techniques, but the computation of a Gröbner basis requires exponential time.

Polynomial decomposition algorithms are directly applicable to Problems 6 and 7, and we will devote Section 3.5 to their solutions.

1.4 An Overview of Results

In Chapter 2, we present the first polynomial time algorithm for the solution to Problem 3, the simple multivariate decomposition problem. This algorithm, which first appeared in [Dic87], is subcubic in the dense representation of the input, and works in any commutative ring containing a multiplicative inverse of $r$, the degree of $g$. We will present the original algorithm of [Dic87], as well as a significant improvement of that algorithm which makes use of the multidimensional discrete Fourier transform (DFT). We will also present two parallel versions of the algorithm: the first is an optimal parallel speedup of the original algorithm, resulting in a linear time parallel algorithm; and the second is a nearly optimal speedup producing a sublinear parallel algorithm. We will also show that the problem is in NC by showing the existence of a parallel version of the algorithm that runs in depth $O(\log^{O(1)} n)$ with polynomially many processors. (Note: by
an optimal parallel speedup, we mean that an $O(T(n))$ time sequential algorithm runs in $O(T(n)/d(n))$ parallel time with $d(n)$ processors.)

In Chapter 3, we present the first polynomial time algorithm for solving Problem 4, computing the left composition factor of a general multivariate composition. We then use this result in the first polynomial time algorithm for Problem 7, computing the inverse of an automorphism. Both algorithms work over any commutative ring and require a number of arithmetic operations which is subcubic in the size of the input and output polynomials in the dense representation. These results first appeared in [Dic88].

In Chapter 4 we show that Problem 5, the s-1-decomposition problem, is NP-hard by reducing the 3-partition problem to it. The 3-partition problem is known to be NP-hard [GJ79]. As Problem 5 is an instance of Problem 1, this serves as proof that the general decomposition problem, Problem 1, is at least NP-hard.

1.5 Notation

Throughout this paper, unless otherwise specified, $K$ denotes a commutative ring with identity. Let $K[x_1, \ldots, x_d]$ be the $d$-variate polynomial ring over $K$. We often abbreviate $x_1, \ldots, x_d$ by $\vec{x}$ and write $K[\vec{x}]$ for $K[x_1, \ldots, x_d]$.

By the “total degree” of $f(\vec{x})$, we mean the sum of the degrees of the $x_i$ in the term (monomial) of $f(\vec{x})$ with the largest such sum. By $\deg_{x_i} f$ we mean the degree of $x_i$ in $f(\vec{x})$ when $f(\vec{x})$ is considered as a polynomial in $K[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n][x_i]$. If we say $\deg f = n_1 \times n_2 \times \cdots \times n_d$ we mean that $\deg_{x_i} f = n_i$. Otherwise, by $\deg f = n$ we mean the total degree of $f$ is $n$. 17
(E.g. if \( f = x^4y^2 + xy^3 \), then \( \deg f = 6 \), \( \deg_x f = 4 \), and \( \deg_y f = 3 \).)

Let \( f(\bar{x}) \) be a polynomial with \( \deg f = n_1 \times n_2 \times \cdots \times n_d \). By "the first \( k_1 \times k_2 \times \cdots \times k_d \) coefficients of \( f(\bar{x}) \)" we mean the coefficients of \( x^{i_1}x^{i_2}\cdots x^{i_d} \) for \( n_j - k_j < i_j \leq n_j, 1 \leq j \leq d \).

A polynomial of degree \( n_1 \times n_2 \times \cdots \times n_d \) is said to be monic if the coefficient of the term \( x^{n_1}x^{n_2}\cdots x^{n_d} \) is 1.

If \( g(\bar{z}) \) and \( h_1(\bar{x}), \ldots, h_d(\bar{x}) \) are polynomials, and \( \bar{z} = z_1, \ldots, z_d \), then by \( g(h(\bar{x})) \) we mean the functional composition \( g(h_1(\bar{x}), \ldots, h_d(\bar{x})) \) of \( g \) and the \( h_i \).

If \( g, h_1, \ldots, h_d \) is a functional decomposition of \( f \), then we call \( g \) the left composition factor and we call the \( h_i \)'s the right composition factors.

Let \( f(\bar{x}) \) and \( g(\bar{x}) \) be polynomials. Then by \( g(\bar{x}) \equiv_i f(\bar{x}) \) we mean that \( g(\bar{x}) \) and \( f(\bar{x}) \) have equal coefficients for all terms of total degree greater than \( i \).

Finally, let \( c^n_i \) denote the \( i^{th} \) elementary symmetric function on an \( n \) element multiset. The \( i^{th} \) elementary symmetric function on an \( n \) element multiset \( A = \{a_1, \ldots, a_n\} \) is given by:

\[
c^n_i(A) = \sum_{\{b_1, \ldots, b_i\} \subseteq A} b_1 \cdots b_i. \tag{1.18}
\]

That is, \( c^n_i(A) \) is the sum of all possible products of exactly \( i \) elements in \( A \).
Chapter 2

Simple Multivariate Decomposition

In this chapter we address the following decomposition problem:

**Problem 3 (Simple Multivariate Decomposition)** Let $K$ be a commutative ring. Given a monic, multivariate polynomial $f(\bar{x}) \in K[\bar{x}]$ of degree $n \times \cdots \times n$ and integers $r$ and $s$ such that $n = rs$ and $r > 1$, decide if there exists a functional decomposition $g, h$ of $f$ with $\deg g = r$ and $\deg h = s \times \cdots \times s$. If so, determine the coefficients of $g$ and $h$.

We will present both sequential and parallel algorithms for simple multivariate decomposition. Our method follows closely the method used by Kozen and Landau [KL86] but with nontrivial generalizations.

The first polynomial time solution to this problem was presented in [Dic87]. It required a number of algebraic operations which was subcubic in the size of the dense representation of the input polynomial $f(\bar{x})$. We begin with the original
algorithm of [Dic87], which we will give in detail for the case $d = 2$ and sketch for the general $d$, and then we give an improved version of that algorithm which uses the Discrete Fourier Transform (DFT). Using a result from [VSBR81], we then show that both algorithms can be parallelized to run in $\log^{O(1)} n$ depth with polynomially many processors. We also give a practical, optimal parallel speedup of the original algorithm, as well as a further near-optimal speedup. All algorithms work over any commutative ring $K$ containing a multiplicative inverse of $r$, the degree of $g$.

2.1 The Basic Algorithm

In this section we give the original multivariate decomposition algorithm from [Dic87] for the case $d = 2$, and an outline of the general multivariate case.

2.1.1 Background and Algorithm: the Case $d = 2$

Given a polynomial $f(x, y) \in K[x, y]$ and an integer $r$, we would like to compute a nontrivial functional decomposition $g, h$ of $f$ with $\deg g = r$.

We begin with some mathematical preliminaries. The following two lemmas are generalizations to the multivariate case of Lemma 1 from [KL86].

**Lemma 2.1** Let $F_1(x, y), F_2(x, y) \in K[x, y]$ be degree $n \times n$ monic polynomials given by:

$$F_1(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{i,j} x^i y^j$$

and

$$F_2(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} b_{i,j} x^i y^j.$$
Let $G(y) \in K[y]$ be a polynomial of degree $m$ such that the products $G(y)F_1(x, y)$ and $G(y)F_2(x, y)$ are given by:

$$G(y)F_1(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m+n} d_{i,j} x^i y^j$$

and

$$G(y)F_2(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m+n} e_{i,j} x^i y^j.$$  

For any $c$, if $a_{c,i} = b_{c,i}$ for $n-k \leq i \leq n$, then $d_{c,i} = e_{c,i}$ for $m+n-k \leq i \leq m+n$. \qed

**Proof** Consider $F_1$ and $F_2$ as polynomials in $K[y][x]$. Let

$$F_1(x, y) = \sum_{i=0}^{n} \alpha_i(y) x^i$$

and let

$$F_2(x, y) = \sum_{i=0}^{n} \beta_i(y) x^i$$

for some $\alpha_0(y), \ldots, \alpha_n(y), \beta_0(y), \ldots, \beta_n(y) \in K[y]$. Then

$$G(y)F_1(x, y) = \sum_{i=0}^{n} G(y) \alpha_i(y) x^i$$

and

$$G(y)F_2(x, y) = \sum_{i=0}^{n} G(y) \beta_i(y) x^i.$$  

By assumption, $\alpha_c(y)$ and $\beta_c(y)$ agree on their first $k+1$ coefficients so by Lemma 1 of [KL86], so do $G(y)\alpha_c(y)$ and $G(y)\beta_c(y)$. But the coefficients of $x^i y^j$ in $G(y)F_1(x, y)$ and $G(y)F_2(x, y)$ are just the coefficients of $y^i$ in $G(y)\alpha_c(y)$ and $G(y)\beta_c(y)$ respectively, and these coefficients agree for $m+n-k \leq i \leq m+n$. \qed
The following more general lemma follows from repeated applications of Lemma 2.1.

**Lemma 2.2** Let $F_1(x, y), F_2(x, y), G(x, y) \in K[x, y]$ be monic polynomials of degree $m \times m$, $m \times m$, and $n \times n$ respectively. If $F_1(x, y)$ and $F_2(x, y)$ agree on their first $I \times J$ coefficients, then so do $G(x, y)F_1(x, y)$ and $G(x, y)F_2(x, y)$. □

**Proof** Consider $G(x, y)$ as a polynomial in $K[y][x]$ and let

$$G(x, y) = \sum_{i=0}^{n} g_i(y)x^i$$

with $g_i(y) \in K[y]$. Then

$$G(x, y)F_1(x, y) = \sum_{i=0}^{n} F_1(x, y)x^ig_i(y) \quad (2.1)$$

and

$$G(x, y)F_2(x, y) = \sum_{i=0}^{n} F_2(x, y)x^ig_i(y). \quad (2.2)$$

By assumption, $F_1(x, y)$ and $F_2(x, y)$ agree on the coefficients of $x^cy^l$ for $m - I < c \leq m$ and $m - J < l \leq m$. Therefore, by switching $x$ and $y$ in Lemma 2.1, we have that $F_1(x, y)x^k$ and $F_2(x, y)x^k$ agree on the coefficients of $x^cy^l$ for $m + k - I < c \leq m + k$ and $m - J < l \leq m$.

Now, also by Lemma 2.1, $F_1(x, y)x^kg_k(y)$ and $F_2(x, y)x^kg_k(y)$ agree on the coefficients of $x^cy^l$ for $m + k - I < c \leq m + k$ and $m + n - J < l \leq m + n$. (Remember, $k$ is the degree of $x^k$ and $m$ is the degree of $g_k(y)$.) Thus for $0 \leq k \leq m$, we have that $F_1(x, y)x^kg_k(y)$ and $F_2(x, y)x^kg_k(y)$ agree (at least) on their coefficients for all terms $x^cy^l$ for $m+n-I < c \leq m+n$, $m+n-J < l \leq m+n$. It is clear that if these terms agree on those coefficients, then so do their sums $G(x, y)F_1(x, y)$ and $G(x, y)F_2(x, y)$. □
Let

\[ f(x, y) = x^n y^n + a_{n-1, n} x^{n-1} y^n + a_{n, n-1} x^n y^{n-1} + \cdots + a_{0, 0} \]

\[ = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{i,j} x^i y^j \in K[x, y] \]

\[ g(x) = x^r + b_{r-1} x^{r-1} + \cdots + b_0 \]

\[ = \sum_{i=0}^{r} b_i x^i \in K(x) \]

\[ h(x, y) = x^s y^s + c_{s-1, s} x^{s-1} y^s + c_{s, s-1} x^s y^{s-1} + \cdots + c_{0, 0} \]

\[ = \sum_{i=0}^{s} \sum_{j=0}^{s} c_{i,j} x^i y^j \in K[x, y] \]

such that \( r, s < n = rs \) and \( f = g \circ h \). Let \( g \) factor in an extension of \( K \) as

\[ g(x) = \prod_{i=1}^{r} (x - \beta_i). \quad (2.3) \]

Then

\[ f(x, y) = g(h(x, y)) = \prod_{i=1}^{r} (h(x, y) - \beta_i). \quad (2.4) \]

By absorbing the constant coefficient of \( h(x) \) into the \( \beta_i \), we can assume without loss of generality that \( c_{0, 0} = 0 \); that is, if \( f = g \circ h \), then let

\[ g_1(x) = \prod_{i=1}^{r} (x - \beta_i + c_{0, 0}) \]

and let

\[ h_1(x, y) = h(x, y) - c_{0, 0}, \]

and we have \( f = g_1 \circ h_1 \). Furthermore, if any normal decomposition \( g, h \) exists for a monic \( f(x, y) \), then there is a normal decomposition \( g', h' \) in which both \( g'(x) \) and \( h'(x, y) \) are also monic. To see this, assume that the lead coefficient \( c_{s, s} \) of \( h(x, y) \) is not 1. We let \( h'(x, y) = h(x, y)/c_{s, s} \) and \( g'(x) = g(c_{s, s} x) \), giving us

\[ g'(h'(x, y)) = g(c_{s, s} h(x, y)/c_{s, s}) = f(x, y). \]

\( h' \) is clearly monic and thus so is
$g'$. Furthermore $c_{s,s}$ is invertible in $K$, since $b_r c_{s,s}^{r} = a_{n,n} = 1$. We can therefore search for a monic, normal decomposition without loss of generality.

By repeated applications of Lemma 2.2 we have:

**Lemma 2.3** The polynomials $h(x, y)^r$ and $f(x, y) = \prod_{i=1}^{r}(h(x, y) - \beta_i)$ agree on their first $s \times (s + 1)$ coefficients and on their first $(s + 1) \times s$ coefficients. That is, they agree on their first $(s + 1) \times (s + 1)$ coefficients with the possible exception of the coefficient of the $x^{r^{s-s}y^{r-s}}$ term. □

**Proof** Our proof is by induction on $r$. For our base case, we take $r = 1$. By inspection, $h(x, y)$ and $h(x, y) - \beta_1$ agree on their first $(s + 1) \times s$ and $s \times (s + 1)$ coefficients, or all coefficients but the constant coefficient.

Assume now that $h(x, y)^i$ and $(h(x, y) - \beta_1) \cdots (h(x, y) - \beta_i)$ agree on their first $(s + 1) \times s$ and their first $s \times (s + 1)$ coefficients for $i \geq 1$. We show that $h(x, y)^{i+1}$ and $(h(x, y) - \beta_1) \cdots (h(x, y) - \beta_{i+1})$ also agree on their first $(s + 1) \times s$ and their first $s \times (s + 1)$ coefficients.

By inductive hypothesis, $h(x, y)^i$ and $(h(x, y) - \beta_1) \cdots (h(x, y) - \beta_i)$ agree on the first $(s + 1) \times s$ and their first $s \times (s + 1)$ coefficients. By Lemma 2.2, so do $h(x, y)^i(h(x, y) - \beta_{i+1})$ and $(h(x, y) - \beta_1) \cdots (h(x, y) - \beta_{i+1})$. But also by Lemma 2.2, so do $h(x, y)^i h(x, y)$ and $h(x, y)^i(h(x, y) - \beta_{i+1})$. Therefore, by transitivity, so do $h(x, y)^{i+1}$ and $(h(x, y) - \beta_1) \cdots (h(x, y) - \beta_{i+1})$. □

We now define new polynomials $q_{j,k}(x, y)$ and $z_{j,k}(x, y)$ which we will use in our algorithm. We define $q_{j,k}(x, y)$ to be the first (highest) $j + 1 \times k + 1$ terms of the polynomial $h(x, y)$.

**Definition 2.1** For $0 \leq j, k \leq s$, we define $q_{j,k}(x, y)$ as:
\[ q_{j,k}(x,y) = \sum_{l=0}^{j} \sum_{m=0}^{k} c_{s-l,s-m} x^{s-l} y^{s-m}. \quad (2.5) \]

That is:

\[ q_{0,0}(x,y) = x^s y^s, \]

and

\[ q_{s,s}(x,y) = h(x,y). \]

Similarly, we define \( z_{j,k} \) to be the first (highest) \( k + 1 \) terms of the polynomial \( \alpha_j(y) x^{s-j} \) when

\[ h(x,y) = \prod_{j=0}^{s} \alpha_j(y) x^{s-j} \]

is viewed as a polynomial in \( K[y][x] \).

**Definition 2.2** For \( 0 \leq j, k \leq s \), we define \( z_{j,k}(x,y) \) as:

\[ z_{j,k}(x,y) = \sum_{i=0}^{k} c_{s-j,s-i} x^{s-j} y^{s-i}. \quad (2.6) \]

From these definitions we can see that:

\[ z_{j,k}(x,y) = z_{j,k-1}(x,y) + c_{s-j,s-k} x^{s-j} y^{s-k}, \quad (2.7) \]

and

\[ z_{j,k}(x,y)^r = \sum_{i=0}^{r} \binom{r}{i} c_{s-j,s-k} x^{s-j} y^{s-k} z_{j,k-1}(x,y)^{r-i}. \quad (2.8) \]
Similarly, we have:

\[ q_{j,k}(x,y) = q_{j-1,k}(x,y) + \sum_{i=0}^{k} c_{s-j,s-i} x^{s-j} y^{s-i} \]
\[ = q_{j-1,k}(x,y) + z_{j,k}(x,y) \]  
(2.9)

Rewriting (2.9) using (2.6) and (2.7) we get:

\[ q_{j,k}(x,y) = q_{j-1,k}(x,y) + z_{j,k-1}(x,y) + c_{s-j,s-k} x^{s-j} y^{s-k} \]  
(2.10)

and

\[ q_{j,k}(x,y)^r = (q_{j-1,k}(x,y) + z_{j,k-1}(x,y) + c_{s-j,s-k} x^{s-j} y^{s-k})^r \]
\[ = q_{j-1,k}(x,y)^r + rq_{j-1,k}(x,y)^{r-1} z_{j,k-1}(x,y) \]
\[ + rq_{j-1,k}(x,y)^{r-1} c_{s-j,s-k} x^{s-j} y^{s-k} + p(x,y) \]  
(2.11)

where \( p(x,y) \) is some polynomial of degree \( \leq rs - 2j \) in \( x \).

We now give some properties of the polynomial \( q_{j,k} \) which we will use in our algorithm. From Definition 2.1 and from repeated applications of Lemma 2.2 we get:

**Lemma 2.4** The polynomials \( h(x,y)^r \) and \( q_{j,k}(x,y)^r \) agree on their first \( (j+1) \times (k+1) \) coefficients for \( 0 \leq j \leq s \) and \( 0 \leq k \leq s \). \( \Box \)

**Proof** Follows exactly the proof technique for Lemma 2.3. \( \Box \)

Lemmas 2.3 and 2.4 directly imply Lemma 2.5.

**Lemma 2.5** The polynomials \( q_{j,k}(x,y)^r \) and \( f(x,y) \) agree on their first \( (j+1) \times (k+1) \) coefficients for \( 0 \leq j \leq s \) and \( 0 \leq k \leq s \), with the possible exception of \( j = k = s \). \( \Box \)
From Lemma 2.5 we see that, for all \(0 \leq j \leq s, 0 \leq k \leq s\) with the possible exception of \(j = k = s\), the coefficient of \(x^{n-j}y^{n-k}\) in \(q_{j,k}(x,y)^r\) equals \(a_{n-j,n-k}\), the coefficient of \(x^{n-j}y^{n-k}\) in \(f(x,y)\). We now relate this equality to the expression for \(q_{j,k}(x,y)^r\) given in (2.11). Let \(d_{j,k}\) be the coefficient of \(x^{n-j}y^{n-k}\) in \(q_{j-1,k}(x,y)^r\). Let \(e_{j,k}\) be the coefficient of \(x^{n-j}y^{n-k}\) in \(q_{j-1,k}(x,y)^{r-1}z_{j,k-1}(x,y)\). Then, from (2.11), we see that the coefficient of \(x^{n-j}y^{n-k}\) in \(q_{j,k}(x,y)^r\) is given by \(d_{j,k} + re_{j,k} + rc_{s-j,s-k}\). This gives us the following equation:

\[
rc_{s-j,s-k} = a_{n-j,n-k} - d_{j,k} - re_{j,k}
\]  

(2.12)

If \(K\) has a multiplicative inverse of \(r\), then from (2.12) we can successively determine the coefficients of \(h\), beginning at \(c_{n,n} = 1\) and working toward \(c_{0,0} = 0\), by solving a series of linear equations given by

\[
c_{s-j,s-k} = \frac{1}{r}[a_{rs-j,rs-k} - d_{j,k} - re_{j,k}].
\]

(2.13)

That is, we begin with \(c_{s,s} = 1\), from which we compute the polynomials \(q_{0,0}(x,y)\) and \(z_{0,0}(x,y)\) and the powers of these polynomials up to the \(r^{th}\) power. We can then compute (simultaneously) \(c_{s,s-1}\) and \(c_{s-1,s}\) using (2.13) as follows:

\[
c_{s,s-1} = \frac{1}{r}(a_{n,n-1} - d_{0,1} - re_{0,1})
\]

\[
c_{s-1,s} = \frac{1}{r}(a_{n-1,n} - d_{1,0} - re_{1,0})
\]

where \(d_{0,1} = 0\) is the coefficient of \(x^{n-1}y^n\) in \(q_{0,0}(x,y)^r\), \(d_{1,0} = 0\) the coefficient of \(x^{n-1}y^n\) in \(q_{0,0}(x,y)^{r-1}z_{0,0}(x,y)\), and \(e_{0,1} = 0\) the coefficient of \(x^{n-1}y^n\) in \(q_{0,0}(x,y)^{r-1}z_{0,0}(x,y)\). Knowing \(c_{s,s-1}\) and \(c_{s-1,s}\), we can now compute the polynomials \(q_{0,1}(x,y), q_{1,0}(x,y), z_{0,1}(x,y)\), and \(z_{1,0}(x,y)\) using (2.10) and (2.8). From there, we proceed to compute (simultaneously) \(c_{s,s-2}, c_{s-1,s-1}\), and \(c_{s,s-2}\), etc.
Algorithm 2.1 (Basic Multivariate Decomposition—Coefficients of $h$)

Step 0.1) Set $q_{0,0}(x, y)^i := x^{is} y^{is}$ for $0 \leq i \leq r$.

Step 0.2) Set $z_{0,0}(x, y)^i := x^{is} y^{is}$ for $0 \leq i \leq r$.

Step $j,k$) $0 \leq j, k \leq s$ and neither $j = k = s$ nor $j = k = 0$:

a) Define $d_{j,k} = \text{the coefficient of } x^{rs-j}s y^{rs-k}$ in $q_{j-1,k}(x, y)^r$.

b) Define $\psi_{j,k,l} = \text{the coefficient of } x^{rs-s}y^{rs-s-l}$ in $q_{j-1,k}(x, y)^{r-1}$.

c) Set $e_{j,k} := \sum_{i=0}^{k-1} c_{s-j,s-i} \psi_{j,k,k-i}$.

d) Set $c_{s-j,s-k} := \frac{1}{r} (a_{rs-j,rs-k} - d_{j,k} - re_{j,k})$.

e) Compute $c_{s-j,s-k}^i$ for $0 \leq i \leq r$.

f) Compute $z_{j,k}(x, y)^i$ for $0 \leq i \leq r$:

$$z_{j,k}(x, y)^i := \sum_{m=0}^{i} \binom{i}{m} (c_{s-j,s-k} x^{s-j} y^{s-k})^m z_{j,k-1}(x, y)^{i-m}.$$  

g) Compute $q_{j,k}(x, y)^i$ for $0 \leq i \leq r$:

$$q_{j,k}(x, y)^i := \sum_{m=0}^{i} \binom{i}{m} z_{j,k}(x, y)^m q_{j-1,k}(x, y)^{i-m}.$$  

\[\square\]

Figure 2.1: Algorithm 2.1

This leads us to Algorithm 2.1 which is given in Figure 2.1.

We make two important observations about this algorithm: in step $(j, k)$, we assume that $q_{j-1,k}(x, y)^0, \ldots, q_{j-1,k}(x, y)^r$ and $z_{j,k-1}(x, y)^0, \ldots, z_{j,k-1}(x, y)^r$ are known from previous steps. If $j < 0$ or $k < 0$ then we let $q_{j,k} = 0$ and $z_{j,k} = 0$. Also, after step $(j, k)$ part (c), we have that $e_{j,k}$ is the coefficient of $x^{rs-j} y^{rs-k}$ in $q_{j-1,k}(x, y)^{r-1} z_{j,k-1}(x, y)$.  

28
Once the coefficients of \( h \) are known, the coefficients of \( g \) may be obtained by solving the triangular system given by:

\[
Ab = a^* \tag{2.14}
\]

where \( A_{ij} \) is the coefficient of \( x^{s_i} y^{s_j} \) in \( h(x, y)^{r-j} \) for \( 0 \leq i, j \leq r \) and \( a^* \) is the vector \([a_{rs,rs}, a_{rs-s,rs-s}, \ldots, a_{0,0}]^T\).

To see this, remember that

\[
f(x, y) = g(h(x, y)) = b_r h(x, y)^r + b_{r-1} h(x, y)^{r-1} + \cdots + b_0
\]

and that \( h(x, y)^i \) is monic of degree \( is \times is \). Thus, letting \( \gamma_{i,j} \) be the coefficient of \( x^{is} y^{js} \) in \( h(x, y)^i \), we see that \( \gamma_{i,j} = 0 \) if \( j > i \) and \( \gamma_{i,j} = 1 \) if \( j = i \). The coefficient \( a_{n,n} \) is given by \( b_r \gamma_{r,r} \); \( a_{n-s,n-s} \) is given by \( b_r \gamma_{r,r-1} + b_{r-1} \gamma_{r-1,r-1} \); and in general:

\[
a_{n-i,n-i} = b_r \gamma_{r,r-i} + b_{r-1} \gamma_{r-1,r-i} + \cdots + b_{r-i} \gamma_{r-i,r-i}. \tag{2.15}
\]

The coefficients of \( g \) and \( h \) are determined uniquely by the above procedure (up to the constant coefficient of \( h \), which we take to be 0). However, the system is overconstrained and the \( g, h \) produced by the above procedure might not compose to give \( f \). We must test to see if \( f(x, y) = g(h(x, y)) \), which we do by computing \( g(h(x, y)) \) directly for the \( h \) and \( g \) produced by the given algorithm.

### 2.1.2 Analysis

Algorithm 2.1 in Figure 2.1 is dominated by substep \( (g) \) of the \((j, k)\) loop, which is iterated \( s^2 \) times. Substep \( (g) \), in turn, is itself a loop which is iterated \( r \) times for each of the \( s^2 \) iterations of the \((j, k)\) loop. In this step, we sum \( r \) terms, each of which requires \( r^3 s^3 \) operations to multiply polynomials of size \( rs \) and \( r^2 s^2 \).
in the dense representation. Algorithm 2.1 therefore requires a total of $O(r^5s^5)$ arithmetic operations.

The number of coefficients of the input polynomial $f(x, y)$ is $N = r^2s^2$ in the dense representation, so we have an $O(N^{2.5})$ time algorithm to compute the coefficients of $h$.

Alternatively, in step (g), we could first compute $q^1_{j,k}$ using the given method, and then compute $q^i_{j,k}$ for $2 \leq i \leq r$ as $q^i_{j,k} := q_{j,k} q^{i-1}_{j,k}$. This latter method would give us an algorithm requiring $O(r^3s^6)$ arithmetic operations. A comparison between these methods clearly depends on the relationship between $r$ and $s$.

To solve for the coefficients of $g$, we solve (2.14). The entries of matrix $A$ are available from Algorithm 2.1. The entries of vector $a^*$ are given as input. The matrix $A$ is easily invertible over $K$ in time $O(r^2)$, since it is triangular with all diagonal elements $1$.

As mentioned, the coefficients of $g$ and $h$ are determined uniquely by the above procedure (up to the constant coefficient of $h$). However, the system is overconstrained and the $g, h$ produced by the above procedure might not compose to give $f$. We can test to see if $f(x, y) = g(h(x, y))$ using the powers of $h$ computed in Algorithm 2.1. We simply compute $h(x, y)^r + b_{r-1}h(x, y)^{r-1} + \cdots + b_0$ and compare the result with $f(x, y)$. There are $r$ polynomials in the above sum, each of size at most $N = n^2$. We can thus compute $g(h(x, y))$ using $O(Nr)$ arithmetic operations.

The overall running time to compute the coefficients of both $g$ and $h$ and to test the solution is thus dominated by Algorithm 2.1 in Figure 2.1.
2.1.3 Generalizing the Algorithm for Arbitrary $d$

In this section, we consider the general multivariate case of $d > 2$. Let $f, g$ and $h$ be as follows:

$$f(x_1, x_2, \ldots, x_d) = \sum_{i_1=0}^{n} \sum_{i_2=0}^{n} \cdots \sum_{i_d=0}^{r_s} a_{i_1, i_2, \ldots, i_d} x_1^{i_1} x_2^{i_2} \cdots x_d^{i_d}$$

$$g(x) = x_r + b_{r-1} x^{r-1} + \cdots + b_0$$

$$h(x_1, x_2, \ldots, x_d) = \sum_{i_1=0}^{s} \sum_{i_2=0}^{s} \cdots \sum_{i_d=0}^{s} c_{i_1, i_2, \ldots, i_d} x_1^{i_1} x_2^{i_2} \cdots x_d^{i_d}$$

where $f(\tilde{x}), h(\tilde{x}) \in K[\tilde{x}]$ and $g(x) \in K[x]$ are monic, and $n = rs$.

One can see how quickly the subscripts get out of hand. For ease of notation, we therefore let $\tilde{i_d} = (i_1, \ldots, i_d)$, $\tilde{j_d} = (j_1, \ldots, j_d)$, $\tilde{n} = (n, \ldots, n)$, $\tilde{s} = (s, \ldots, s)$ and $\tilde{0} = (0, \ldots, 0)$. Then, for instance, $\tilde{s} - \tilde{i_d}$ would be $(s - i_1, s - i_2, \ldots, s - i_d)$. We can now rewrite $f, g$ and $h$ as:

$$f(\tilde{x}) = \sum_{\tilde{i_d} = \tilde{0}}^{\tilde{n}} a_{\tilde{i_d}} x^{\tilde{i_d}}$$

$$g(x) = x_r + b_{r-1} x^{r-1} + \cdots + b_0$$

$$h(\tilde{x}) = \sum_{\tilde{i_d} = \tilde{0}}^{\tilde{s}} c_{\tilde{i_d}} x^{\tilde{i_d}}$$

Assume $f(\tilde{x}) = g(h(\tilde{x}))$. Given $r$, the degree of $g$, we want to decompose $f$ to find $g$ and $h$.

We begin with a definition.

**Definition 2.3** We define $Z_{k,j_d}(\tilde{x})$ as follows:

$$Z_{k,j_d}(\tilde{x}) = \sum_{\tilde{i_k} = \tilde{0}}^{\tilde{j_k}} c_{s-i_1, \ldots, s-i_k, s-j_k+1, \ldots, s-j_d} x_1^{s-i_1} \cdots x_k^{s-i_k} x_{k+1}^{s-j_k+1} \cdots x_d^{s-j_d} \quad (2.16)$$

$\square$
Then
\[
Z_{0,j_d}^r(\vec{x}) = c_{s-j_1,s-j_2,\ldots,s-j_d}x_1^{s-j_1}x_2^{s-j_2} \ldots x_d^{s-j_d}
\]
\[
Z_{1,j_d}^r(\vec{x}) = \sum_{i_1=0}^{j_1} c_{s-i_1,s-j_2,\ldots,s-j_d}x_1^{s-i_1}x_2^{s-j_2} \ldots x_d^{s-j_d}
\]
\[\vdots\]
\[
Z_{d,j_d}^r(\vec{x}) = \sum_{i_d=0}^{j_d} c_{s-i_1,s-i_2,\ldots,s-i_d}x_1^{s-i_1}x_2^{s-i_2} \ldots x_d^{s-i_d}
\]
\[
= \sum_{i_d=0}^{j_d} c_{s-i_1,s-j_2,\ldots,s-j_d}x_1^{s-i_1}x_2^{s-i_2} \ldots x_d^{s-i_d}
\]

That is: \(Z_{0,j_d}^r\) is a single term from \(h\); \(Z_{k,j_d}^r\) is a \(k\)-dimensional array of terms from \(h\); \(Z_{d,j_d}^r\) consists of the first \(j_1 \times j_2 \times \cdots \times j_d\) terms of \(h\); and \(Z_{d,s} = h\). (\(Z_{d,j_d}^r\) thus replaces \(q\) in Algorithm 2.1.)

We can show as a result of Lemmas 2.6 and 2.7 below that \(Z_{d,j_d}^r, h^r,\) and \(f\) agree on their first \(j_1 \times j_2 \times \cdots \times j_d\) coefficients for \(0 \leq j_i \leq n\) with the possible exception of \(j_d = s\). Thus, at step \((j_d')\) of the general algorithm, we can compute \(c_{s-j_1,s-j_2,\ldots,s-j_d}\) from the previously computed coefficients.

**Lemma 2.6** Given polynomials \(F_1(\vec{x}), F_2(\vec{x}), G(\vec{x}) \in K[\vec{x}]\) of degree \(\underbrace{d \times \cdots \times d}_{m \times \cdots \times m}\), and \(\underbrace{n \times \cdots \times n}_{m \times \cdots \times m}\) respectively, all of dimension \(d \geq 2\), if the polynomials \(F_1(\vec{x})\) and \(F_2(\vec{x})\) agree on their first \(j_1 \times j_2 \times \cdots \times j_d\) coefficients for \(j_1, j_2, \ldots, j_d \leq m\), then so do the products \(G(\vec{x})F_1(\vec{x})\) and \(G(\vec{x})F_2(\vec{x})\).

**Proof** The proof is by induction on \(d\), the dimension of \(K[\vec{x}]\). The base case, \(d = 2\), is given by Lemma 2.5. For the inductive hypothesis, assume the lemma is true for \(d - 1\). We will show it true for \(d\).
Consider the polynomials \( F_1(\bar{x}), F_2(\bar{x}), G(\bar{x}) \in K[\bar{x}] \) to be polynomials in \( K[x_2, \ldots, x_d][x_1] \). Let

\[
F_1(\bar{x}) = \sum_{j=0}^{m} \alpha_j(x_2, \ldots, x_d)x_1^j \\
= \sum_{j=0}^{m} \alpha_j x_1^j
\]

\[
F_2(\bar{x}) = \sum_{j=0}^{m} \beta_j(x_2, \ldots, x_d)x_1^j \\
= \sum_{j=0}^{m} \beta_j x_1^j
\]

\[
G(\bar{x}) = \sum_{i=0}^{n} g_i(x_2, \ldots, x_d)x_1^i \\
= \sum_{i=0}^{n} g_i x_1^i
\]

with \( \alpha_i, \beta_i, g_i \in K[x_2, \ldots, x_d] \) for \( 0 \leq j \leq m \) and \( 0 \leq i \leq n \). Then we have

\[
G(\bar{x})F_1(\bar{x}) = \sum_{i=0}^{n} g_i x_1^i \sum_{j=0}^{m} \alpha_j x_1^j \\
= \sum_{i=0}^{n} \sum_{j=0}^{m} g_i \alpha_j x_1^{i+j}
\]

and

\[
G(\bar{x})F_2(\bar{x}) = \sum_{i=0}^{n} g_i x_1^i \sum_{j=0}^{m} \beta_j x_1^j \\
= \sum_{i=0}^{n} \sum_{j=0}^{m} g_i \beta_j x_1^{i+j}
\]

By assumption, the polynomials \( \alpha_j \) and \( \beta_j \) of dimension \( d - 1 \) agree on their first \( j_2 \times \cdots \times j_d \) coefficients for \( m - j_1 < j \leq m \). Therefore, by our inductive
hypothesis, the $d - 1$ dimension polynomials $g_i \alpha_j$ and $g_i \beta_j$ also agree on their first $j_2 \times \cdots \times j_d$ coefficients for $m - j_1 < j \leq m$.

It is a simple step from there to show that the $d$ dimension polynomials $g_i \alpha_j x_1^{i+j}$ and $g_i \beta_j x_1^{i+j}$ also agree on their first $1 \times j_2 \times \cdots \times j_d$ coefficients for $m - j_1 < j \leq m$. It follows that the sums of these terms, namely $G(\bar{x})F_1(\bar{x})$ and $G(\bar{x})F_2(\bar{x})$, agree on their first $j_1 \times \cdots \times j_d$ coefficients. \qed

**Lemma 2.7** The polynomials $Z_{d,j_d}^i$, $h^r$, and $f$ agree on their first $j_1 \times j_2 \times \cdots \times j_d$ coefficients for all $0 \leq j_i \leq n$ with the possible exception of $j_d = \bar{s}$.

**Proof** The proof follows exactly the method of the proof of Lemma 2.3, using Lemma 2.6. \qed

Now note the following equivalence, which we can use to compute the $Z_{i,j_d}$:

$$Z_{i,j_d} = Z_{i,j_1,j_2,\ldots,j_{i-1},j_d} + Z_{i-1,j_d}.$$  \hspace{2cm} (2.17)

Using (2.17), we can compute all of the $Z_{i,j_d}^j$ from the $Z_{i-1,j_d}^j$ computed during the same step and from the $Z_{i,j_d}^j$ computed during the previous steps, as in steps (f) and (g) of Algorithm 2.1. Alternatively, we could compute $Z_{i,j_d}^1$ using this method and then compute $Z_{i,j_d}^j$ for $2 \leq j \leq r$ as $Z_{i,j_d}^j := Z_{i,j_d}^1 Z_{i,j_d}^{j-1}$. These two methods lead us to general decompositon algorithms for arbitrary $d$ requiring $O(r^{2d+1}s^{3d-1})$ and $O(r^{d+1}s^{3d})$ arithmetic operations, respectively. The number of coefficients of $f$ is $N = r^d s^d$, so we have algorithms which are $O(N^3)$, where $N$ is the size of the input polynomial $f$ in the dense representation.
2.1.4 Decomposing Non-monic Polynomials

The techniques in this chapter apply to the decomposition of monic polynomials \( f(\vec{x}) \). If the lead coefficient (the coefficient of \( x^n \cdots x^a \)) is not 1, the techniques still may apply.

Let

\[
f(\vec{x}) = \sum_{\vec{i} \in \mathbb{N}^d} a_{\vec{i}} \vec{x}^{\vec{i}}
\]

with \( a_{n,\ldots,n} \neq 1 \). If \( a_{n,\ldots,n} \) is a unit in \( K \), then let \( f'(\vec{x}) = f(\vec{x})/a_{\vec{i}} \) be a monic polynomial in \( K[\vec{x}] \). We can now use our techniques for monic polynomials to decompose \( f' \). If \( g', h \) is a normal decomposition of \( f' \), then clearly \( g, h \) is a normal decomposition of \( f \), where \( g(x) = a_{\vec{i}}g'(x) \).

We can thus compute decomposition of non-monic polynomials \( f(\vec{x}) \) provided \( a_{\vec{i}} \) has an inverse in \( K \).

2.2 A Speedup using Discrete Fourier Transforms

In this section, we will show how to speed up the Basic Multivariate Polynomial Decomposition Algorithm 2.1 of Figure 2.1 significantly. The method works whenever \( K \) supports a multivariate discrete Fourier transform (DFT).

For ease of notation, we will again consider the case \( d = 2 \) and leave the generalization to larger values of \( d \) to the reader. We show that the \( O(r^5 s^5) \) sequential bound for the case \( d = 2 \) can be improved to \( O(r^2 s^4 \log r) \) using a multidimensional DFT, provided appropriate roots of unity are present. (It has also been pointed out by Dexter Kozen that the same technique of executing the
algorithm in transformed space can be used even when a Fourier transform is not supported. This observation allowed for some of the improvements from [KL86] to [KL89], independent of the DFT. A general transform using values other than the roots of unity, however, is not as efficient as a Fourier transform, and would produce an $O(r^4s^4)$ or $O(N^2)$ algorithm.

For general $d$, the bound is improved from $O(r^{2d+1}s^{3d-1})$ to $O(r^ds^{2d}\log r)$ or from $O(N^3)$ to $O(s^dN\log n)$.

2.2.1 Background and Algorithm

Assume there exists $m \geq n + 1, m = O(n)$, such that $K$ contains a principal $m^{th}$ root of unity $\zeta$ and a multiplicative inverse of $m$. The two-dimensional Fourier transform $F(f)$ of the polynomial $f(x, y)$ can be represented as the vector:

$$(f(1, 1), f(1, \zeta), \ldots, f(1, \zeta^{m-1}), f(\zeta, 1), f(\zeta, \zeta), \ldots, f(\zeta^{m-1}, \zeta^{m-1}))^T.$$  

The multidimensional discrete Fourier transform requires $O(N\log n)$ operations when $m$ is prime and $O(dN\log n)$ operations otherwise [AFW83, NQ79].

Multiplication of a transformed polynomial is computed pointwise, and therefore requires $O(m^d)$ (or $O(N)$) operations rather than $O(n^{2d})$ (or $O(N^d)$) operations for $d$-variate polynomials. We can thus compute the product $G \cdot H$ of two polynomials $G$ and $H$ more efficiently, as follows:

$$G \cdot H := F^{-1}(F(G) * F(H))$$  \hspace{1cm} (2.19)

where $*$ denotes pointwise multiplication of vectors. This gives us an $O(dN\log n)$ algorithm for computing the product of two polynomials. This method can be generalized to compute the product of an arbitrary number of polynomials. We use this method to improve Algorithm 2.1.
Algorithm 2.2 in Figure 2.2 is equivalent to Algorithm 2.1 in Figure 2.1 but is carried out in transformed space.

(Three observations should be made about this algorithm. In step \((j, k)\) of the algorithm, we assume that \(e_{j-1,k}\) and \(w_{j,k-1}\) are known from previous steps. Also, if \(j < 0\) or \(k < 0\), let \(e_{j,k} = F(0)\) and \(w_{j,k} = F(0)\). Finally, all products and powers in Fourier space are computed pointwise.)

### 2.2.2 Analysis

The Fourier transforms in steps 0 and \((s,s)\) require \(O(N \log n)\) arithmetic operations. In substep \((e)\), however, all of \(F^{-1}(\tilde{e})\) need not be computed, but only the appropriate coefficient. Likewise, computing \(F(cx^{s-jy^{s-k}})\) requires fewer operations than a full Fourier transform. Substep \((e)\) can be performed with only \(O(N)\) arithmetic operations per iteration. Substep \((a)\) therefore dominates the \((j,k)\) loop, requiring \(O(N \log r)\) operations per iteration of the loop. The entire algorithm therefore requires only \(O(s^2 N \log r)\) arithmetic operations. This is a significant improvement over the \(O(N^{2.5})\) running time of Algorithm 2.1. For general \(d\), the running time of this algorithm is \(O(s^d N \log r)\).

### 2.3 Parallel Algorithms

In this section, we will show that both Algorithms 2.1 and 2.2 can be parallelized so as to run in \(O(\log^{O(1)} N)\) depth with polynomially many processors. In addition to this theoretical result, we will also show a practical, optimal parallel speedup of the original algorithm which results in a linear-time parallel algorithm, as well as another parallel speedup which is nearly optimal and results in
Algorithm 2.2 (Multivariate Decomposition with DFT—Coefficients of \( h \))

Step 0.1) Set \( \hat{f} := F(f) \), the Fourier transform vector of \( f \).

Step 0.2) Set \( e_{0,0} := F(q_{0,0}) \) where \( q_{0,0}(x,y) = x^s y^s \).

Step 0.3) Set \( w_{0,0} := F(z_{0,0}) \) where \( z_{0,0}(x,y) = x^s y^s \).

Step \( j,k \) for \( 0 \leq j,k \leq s \) and neither \( j = k = 0 \) nor \( j = k = s \):

a) Set \( \bar{e} := e_{j-1,k}^{r-1} \).
   
   ( Now \( \bar{e} = F(q_{j-1,k}^{r-1}) \). )

b) Set \( \bar{w} := r(w_{j,k-1} \bar{e}) \).
   
   ( Now \( \bar{w} = F(rz_{j,k-1}q_{j-1,k}^{r-1}) \). )

c) Set \( \bar{e} := \bar{e}e_{j-1,k} \).
   
   ( Now \( \bar{e} = F(q_{j-1,k}^{r-1}) \). )

d) Set \( \bar{e} := \frac{1}{r}(\bar{f} - \bar{e} - \bar{w}) \).

e) Set \( c := \text{the coefficient of } x^{r s-j} y^{s-k} \text{ in } F^{-1}(\bar{e}) \).

Set \( e := F(cx^{s-j} y^{s-k}) \).

f) Set \( w_{j,k} := w_{j,k-1} + \bar{e} \).

g) Set \( e_{j,k} := e_{j-1,k} + w_{j,k} \).

Step \( s,s \) Set \( h(x,y) := F^{-1}(e_{s-1,1} + w_{s,s-1}) \).

\( \square \)

Figure 2.2: Algorithm 2.2
a sublinear parallel algorithm.

2.3.1 NC Result

Both previous sequential algorithms produce expressions for the coefficients of \( g \) and \( h \) of degree at most \( r \) in each coefficient of \( f \). Since these coefficients are calculated by a straight-line program involving only algebraic operations, and since the Fourier transforms in steps 0 and \( s, s \) of Algorithm 2.2 are in NC, it follows from a general result of [VSBR81] that the algorithms can be parallelized so as to run in \( \log^{O(1)} N \) parallel time, using polynomially many processors.

2.3.2 Practical Parallel Speedups

In addition to this theoretical result showing the existence of NC algorithms, there is also a practical, optimal speedup of Algorithm 2.1 which uses a small number of processors, \( O(N^{1.5}) \), and produces a parallel algorithm with linear running time.

Consider substep (g) of Algorithm 2.1. (Substep (g) dominates the algorithm.) In substep (g) we calculate the following:

\[
q_{j,k}(x, y)^i := \sum_{m=0}^{i} \binom{i}{m} z_{j,k}(x, y)^m q_{j-1,k}(x, y)^{i-m}
\]  

(2.20)

The product \( z_{j,k}(x, y)^m q_{j-1,k}(x, y)^{i-m} \) has at most \( r^2 s^2 \) coefficients. Given \( r^2 s^2 \) processors, assign one processor to each term in the product. Each processor will multiply each of the at most \( rs \) coefficients of \( z_{j,k}(x, y)^m \) by at most one coefficient of \( q_{j-1,k}(x, y)^{i-m} \) such that the product of the two contributes to the specified coefficient in the product. Thus, with \( r^2 s^2 \) processors, we have reduced the running time for computing the product \( z_{j,k}(x, y)^{i-m} q_{j-1,k}(x, y)^{i-m} \) from
$O(r^3s^3)$ to $O(rs)$. With $N$ processors, we have a factor of $N$ speedup. The same technique applies to substep (f) as well.

Given $r$ times as many processors, we can compute each of the $q_{j,k}(x,y)^i$ terms in substeps (f) and (g) for $0 \leq i \leq r$ in parallel, since the higher powers are not computed using the values of the lower powers.

Finally, note that for any constant $C$, all steps $j,k$ for $j + k = C$ can be computed in parallel, provided that steps $j,k$ for $j + k < C$ have all been computed previously. Thus with $s$ times as many processors, we can improve the running time by a factor of $s/2$ by executing steps $j,k$ for $j + k = C$ in parallel for $C$ running from 0 to $s/2$.

Using all three of the above techniques, we have an $O(N^{1.5})$ factor speedup using $N^{1.5}$ processors, yielding an $O(N)$ algorithm. The above techniques work for general $d$ to yield a $\frac{1}{d}Nrs^{d-1}$ factor speedup using $Nrs^{d-1}$ processors. The parallel running time for general $d$ using these techniques is $O(Nd)$.

The parallel algorithm can be further improved to sublinear time using a small factor of additional processors and a near-optimal speedup. Consider substep (g) again. Instead of assigning a single processor, assign $rs$ processors to each coefficient in the product $z_{j,k}(x,y)^m q_{j-1,k}(x,y)^{i-m}$. Each processor, in constant time, computes one of the $rs$ terms contributing to the sum. The $rs$ terms can then be summed in $O(\log(rs))$ time using those $rs$ processors. This gives an $O(rs/\log(rs))$ factor speedup using $rs$ processors.

Finally, with an additional factor of $r$ processors, we can compute each of the products $\binom{i}{m} z_{j,k}(x,y)^m q_{j-1,k}(x,y)^{i-m}$ for $0 \leq m \leq i$ and $i \leq r$ in parallel and add them in $O(\log r)$ time, giving us an $O(r/\log r)$ factor speedup.
using $r$ processors. Using all of the above methods, we have a nearly optimal speedup of $O(r^5s^4/\log r \log(rs))$ using $r^5s^4$ processors. The final result is an $O(s \log r \log(rs))$-depth parallel algorithm requiring $r^5s^4$ processors, or for arbitrary $d$, an $O(d^2s \log(rs))$-depth parallel algorithm requiring $r^{2d+1}s^{3d-2}$ processors.

Using the second method of computing $Z_i^i$ as $Z_i^i := Z_i Z_i^{i-1}$ and similar parallelization techniques gives us an $O(d^2r s \log s)$-depth algorithm using only $r d s^{3d-1}$ processors.

Letting $N = r^d s^d$ be the size of the input in the dense representation, we have in both cases an $O(d^2 \sqrt[N]{N} \log N)$-depth parallel algorithm requiring $N^3$ processors.

### 2.4 Other Results

In October 1987, von zur Gathen [vzG87] presented an algorithm to solve Problem 3 which requires only $O(ndN \log n)$ operations, assuming that the field $K$ supports a Fourier transform. Depending on the relationship between $r$ and $s$, his algorithm may be more or less favorable than Algorithm 2.2 presented here. Both algorithms work only in the “tame case” where the characteristic of the field does not divide $r$, and both require that $K$ support a Fourier transform.

The basic idea behind von zur Gathen’s method is interesting and worth noting. The method involves a reduction of the multivariate case to the univariate case in order to solve for the coefficients $g$. To solve for $g$, we note the following: Let $f(\vec{x})$ have a decomposition $g, h$ with $h(\vec{0}) = 0$. Define $f'(x_1) = f(x_1, 0, \ldots, 0)$ and define $h^*(x_1) = h(x_1, 0, \ldots, 0)$. Let the univariate polynomial
$f'(x_1)$ decompose as $f'(x_1) = g'(h'(x_1))$ with $h'(0) = 0$. Since $f(x_1, 0, \ldots, 0) = g(h(x_1, 0, \ldots, 0))$, it is clear that $f'(x_1) = g(h^*(x_1))$, and since the normal decomposition of $f'$ for a given $r = \deg g$ is unique, we have $g = g'$ and $h^* = h'$. We can therefore compute $g$ by reducing $f(\bar{x})$ to $f'(x_1)$ and decomposing the univariate polynomial $f'(x_1)$. From there, von zur Gathen uses a linearly converging Newton method to compute $h$ from $h'$. For details, see [vzG88].
Chapter 3

Left Composition Factor of a Multivariate Decomposition

In this chapter, we address the following problem:

Problem 4 (Multivariate Left Composition) Given polynomials $f(\vec{x})$ and $h_1(\vec{x}), \ldots, h_d(\vec{x}) \in K[\vec{x}]$, $\vec{x} = x_1, \ldots, x_d)$, and an integer $r$, decide if there exists a polynomial $g(\vec{x}) \in K[\vec{x}]$ of total degree at most $r$ that composes with the $h$'s to give $f$. That is, does there exist a polynomial $g(\vec{x})$ such that $f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_d(\vec{x}))$ and $\deg g \leq r$? If so, determine the coefficients of $g$.

We present the first polynomial time algorithm for Problem 4, which we call the problem of computing the left composition factor in a multivariate polynomial decomposition. We then use that result in the first polynomial time algorithm for computing the inverse of an automorphism of a polynomial ring. Finally, we show how these algorithms can be used to determine in exponential time whether an endomorphism over a polynomial ring is an automorphism.
3.1 Computing the Left Composition Factor

In this section, we present a polynomial time algorithm for computing the left composition factor in a multivariate polynomial decomposition. This is a special case of Problem 1. We will give the details for the case \( d = 2 \) (the bivariate case) and then the results for the case of arbitrary dimension \( d \) (the general multivariate case).

3.1.1 Background and Algorithm: the Case \( d = 2 \)

We begin with the case \( d = 2 \). We examine the following bivariate version of Problem 4:

**Problem 9** Given polynomials \( f(x, y), h_1(x, y) \) and \( h_2(x, y) \in K[x, y] \) of degrees \( n, s, \) and \( s \) respectively, and an integer \( r \), decide whether there exists a polynomial \( g(z_1, z_2) \in K[z_1, z_2] \) such that

\[
f(x, y) = g(h_1(x, y), h_2(x, y))
\]

and \( \deg g \leq r \). If so, compute the coefficients of \( g \).

Assume \( f(x, y) = g(h_1(x, y), h_2(x, y)) \) for some \( f, g, h_1, h_2 \in K[x, y] \) of total degrees \( n, r, s \) and \( s \) respectively. It is clear that \( rs \geq n \). Let

\[
g(z_1, z_2) = \sum_{i=0}^{r} \sum_{j=0}^{r-i} b_{i,j} z_1^i z_2^j. \tag{3.1}
\]

Then

\[
f(x, y) = \sum_{i=0}^{r} \sum_{j=0}^{r-i} b_{i,j} h_1(x, y)^i h_2(x, y)^j. \tag{3.2}
\]

It is easy to show from (3.2) that
\[ f(x, y) \equiv_R \sum_{i=0}^{r} b_{i,r-i} h_1(x, y)^i h_2(x, y)^{r-i} \quad (3.3) \]

where \( R = rs - s \). That is, \( f(x, y) \) and \( \sum_{i=0}^{r} b_{i,r-i} h_1(x, y)^i h_2(x, y)^{r-i} \) agree on their coefficients for all terms of total degree greater than \( rs - s \).

More generally, defining \( b_{i,j} = 0 \) for \( i < 0 \) or \( j < 0 \), we can show that for \( 0 \leq k \leq r \) we have:

\[
f(x, y) \equiv_R \sum_{i=0}^{r} \sum_{j=r-i-k}^{r-i} b_{i,j} h_1(x, y)^i h_2(x, y)^j \quad (3.4)
\]

where \( R = rs - s - ks \). Intuitively, we are using the fact that only those products \( h(x, y)^i h(x, y)^j \) with \( i + j > c \) can produce terms of degree greater than \( cs \) to show that only those coefficients \( b_{i,j} \) of \( g \) where \( i + j > c \) affect the coefficients of \( f \) of degree greater than \( cs \).

It follows from (3.4) that the degree of

\[
f(x, y) - \sum_{i=0}^{r} \sum_{j=r-i-k}^{r-i} b_{i,j} h_1(x, y)^i h_2(x, y)^j
\]

is at most \( rs - s - ks \). Subtracting

\[
\sum_{i=0}^{r} \sum_{j=r-i-(k-1)}^{r-i} b_{i,j} h_1(x, y)^i h_2(x, y)^j
\]

from both sides of (3.4) we get:

\[
f(x, y) - \sum_{i=0}^{r} \sum_{j=r-i-(k-1)}^{r-i} b_{i,j} h_1(x, y)^i h_2(x, y)^j
\]

\[
\equiv_R \sum_{i=0}^{r} b_{i,r-i-k} h_1(x, y)^i h_2(x, y)^{r-i-k} \quad (3.5)
\]

where \( R = rs - s - ks \). Equation 3.5 gives us a recurrence relation for the \( b_{i,j}'s \).

We can compute the coefficients \( b_{i,r-i-k} \) for \( 0 \leq i \leq r - k \) in terms of \( b_{i,r-i-j} \) for \( 0 \leq i \leq r - j, j < k \). For example, we use (3.3) to compute \( b_{i,j} \) for \( i + j = r \).
Then, by substituting 1 for $k$ in (3.5) we get:

$$f(x, y) - \sum_{i=0}^{r} b_{i,r-i} h_1(x, y)^i h_2(x, y)^{r-i}$$

$$\equiv_R \sum_{i=0}^{r} b_{i,r-i} h_1(x, y)^i h_2(x, y)^{r-i-1}$$  \hspace{1cm} (3.6)$$

where $R = rs - 2s$. We can use these values of the coefficients $b_{i,j}$ for $i + j = r$ to compute the coefficients $b_{i,j}$ for $i + j = r - 1$.

This is exactly what we do in Algorithm 3.1 given in Figure 3.1.

### 3.1.2 An Explanation

Algorithm 1 hinges on solving the system of equations in step (2) substep (b) of the algorithm. Let us see when and how we are able to do this.

First note that we have a system of $(r - k - 1)s^2 + (s^2 + s)/2$ linear equations in the $r + 1 - k$ unknowns $b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}$. Our system is given by the equation:

$$C_k b_k = a_k$$  \hspace{1cm} (3.7)$$

where we define $C_k, b_k$, and $a_k$ as in Figure 3.2. The system given by Equation 3.7 is overdetermined, but may or may not be rank deficient. In step (2) substep (b) we may get no solution, a unique solution, or a solution space of dimension $\leq r - k$.

It is clear from (3.5) that if there is no solution to (3.7), then no decomposition exists for the given $f, h_1, h_2$. In this case, we can halt and output "NO DECOMPOSITION".

If we get a unique solution, then we continue the algorithm exactly as written.
Algorithm 3.1 Left Composition Factor

**Step 0)** For $0 \leq i \leq r$

For $0 \leq j \leq r - i$

Compute $h_1(x, y)^i h_2(x, y)^j$.

Let $c_{i,j,k,l}$ be the coefficient of $x^k y^l$ in $h_1(x, y)^i h_2(x, y)^j$.

**Step 1)** Let $f_0(x, y) := f(x, y)$.

**Step 2)** For $k := 0$ to $r - 1$ do:

a) Let $a_{k,i,j}$ be the coefficient of $x^i y^j$ in $f_k(x, y)$.

b) Solve the system of equations given by:

$$
\sum_{l=0}^{r-k} b_{l,r-l-k} c_{l,r-l-k,i,j} = a_{k,i,j}
$$

for $rs - s - ks < i + j \leq rs - ks$. (See Figure 3.2)

c) Let $f_{k+1}(x, y) := f_k(x, y) - \sum_{l=0}^{r-k} b_{l,r-l-k} h_1(x, y)^l h_2(x, y)^{r-l-k}$.

**Step 3)** Let $b_{0,0} := f_r(x, y)$. ($f_r$ is a constant.)

□

Figure 3.1: Algorithm 3.1

47
Figure 3.2: $C_k b_k = a_k$
The interesting and difficult case is when, at step \((2.k)\) substep \((b)\), we get a solution space of positive dimension. When our solution to this step is of positive dimension, then in substep \((c)\) we will get an expression for \(f_{k+1}\) in terms of \(f_k\) and the unknowns \(b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}\). Thus, during the subsequent step, step \((2.k + 1)\), we must solve for \(b_{0,r-(k+1)}, b_{1,r-(k+1)-1}, \ldots, b_{r-(k+1),0}\) in terms of \(b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}\). We look now at the equation \(C_{k+1}b_{k+1} = a_{k+1}\). Once again, the matrix \(C_{k+1}\) may or may not be of full rank. If it is of full rank, then choose \(r-k\) linearly independent rows and call these rows \(C'_{k+1}\). The corresponding entries in \(a_{k+1}\) we call \(a'_{k+1}\). We then solve the system \(C'_{k+1}b_{k+1} = a'_{k+1}\) to get a solution for the coefficients \(b_{0,r-(k+1)}, b_{1,r-(k+1)-1}, \ldots, b_{r-(k+1),0}\) in terms of \(b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}\). We can then use the remaining equations in the system \(C_{k+1}b_{k+1} = a_{k+1}\) to solve for the coefficients \(b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}\), thus reducing the dimension of the solution space.

If the system \(C_{k+1}b_{k+1} = a_{k+1}\) is also rank deficient, then when we solve for the \(b_{i,r-(k+1)-i}\), the dimension of our current solution space will increase, but only by at most \(r-k-1\). The important thing to note is that in the case when we have a solution space of positive dimension, then the \(a_{k,i,j}\) in the vector \(a_k\) at step \((2.k)\) are given by linear equations. These linear equations, and thus the dimension of the solution space as well, can grow by at most \(r-k\) linear terms or \(O(r)\) at each step. Therefore these equations for \(a_{k,i,j}\) reach a size of at most \(O(r^2)\) in the worst case.

### 3.1.3 Analysis

We now give an analysis if the number of arithmetic operations required by Algorithm 3.1. The algorithm is clearly dominated by steps \((0)\) and \((2)\), so we
will look at those in detail.

**Step (0)** A naive approach to step (0) would be to compute all of the powers \( h_1(x, y)^i \) and \( h_2(x, y)^i \) for \( 1 \leq i \leq r \) and to compute \( h_1(x, y)^i h_2(x, y)^j \) from those. This approach would require \( O(r^6s^4) \) arithmetic operations. An asymptotically faster approach would be to compute \( h_1(x, y)^i \) for \( 1 \leq i \leq r \) and then to compute \( h_1(x, y)^i h_2(x, y)^j \) for \( 1 \leq j \leq r \) as

\[
h_1(x, y)^i h_2(x, y)^j := h_2(x, y)(h_1(x, y)^i h_2(x, y)^{j-1})
\]  

(3.8)

This latter method requires only \( O(r^4s^4) \) arithmetic operations.

If \( K \) supports a multivariate discrete Fourier transform, then we can improve our results further by transforming the polynomials into Fourier space, performing the same operations, and then doing the inverse transform on the result. The multivariate DFT and its inverse each require \( O(n^2 \log n) \) time for a bivariate polynomial of degree \( n \) [AFW83,NQ79]. Multiplication is performed pointwise on a transformed vector and thus requires only \( O(n^2) \) arithmetic operations on a bivariate polynomial of degree \( n \). Computing the required polynomials \( h_1(x, y)^i h_2(x, y)^j \) would therefore require \( O(r^3s^2) \) operations in transformed space and the \( r^2 \) inverse transforms would each require \( O(r^2s^2 \log(rs)) \) time. Step (0) therefore requires \( O(r^4s^2 \log(rs)) \) arithmetic operations when a multivariate DFT is used.

**Step (2)** The system of equations in step (2) substep (b) has size

\[
[(r - k + 1/2)s^2 + s/2] \times [r + 1 - k]
\]
which is \( O(r^2 s^2) \times O(r) \). The \( a_{k,i,j} \) are given by linear equations of at most \( O(r^2) \) terms in the worst case. The system can therefore be solved with \( O(r^4 + r^3 s^2) \) arithmetic operations using the techniques of [GvL83] applied symbolically. It therefore requires \( O(r^5 + r^4 s^2) \) arithmetic operations to perform \( r \) iterations of step (2).

The entire algorithm requires \( O(r^5 + r^4 s^2 \log(rs)) \) arithmetic operations. Let \( I = n^2 + s^2 \) be the size of the input and \( N = r^2 \) be the size of the output, both in the dense representation. At worst our algorithm requires \( O(I^{2.5} + I^2 N \log(IN)) \) or \( O((I + N)^{2.5}) \) operations.

When \( r \) is small, the algorithm is nearly linear in the size of the input.

### 3.1.4 Generalizing the Algorithm to Arbitrary \( d \)

The step from the bivariate case to the general multivariate case is quite simple.

Let \( f(\bar{x}) = g(\bar{h}(\bar{x})) \) and let \( g(\bar{z}) \) be defined as follows:

\[
g(\bar{z}) = \sum_{i_k \geq 0, 1 \leq k \leq d} b_{i_1, i_2, \ldots, i_d} z_1^{i_1} z_2^{i_2} \cdots z_d^{i_d}. \tag{3.9}
\]

\[
\begin{align*}
&i_k \geq 0, 1 \leq k \leq d \\
&i_1 + \cdots + i_d \leq r
\end{align*}
\]

We now give the multivariate analog to (3.4):

\[
f \equiv_R \sum_{i_k \geq 0, 1 \leq k \leq d} b_{i_1, i_2, \ldots, i_d} h_1^{i_1} h_2^{i_2} \cdots h_d^{i_d} \tag{3.10}
\]

\[
\begin{align*}
&i_k \geq 0, 1 \leq k \leq d \\
&r - k \leq i_1 + \cdots + i_d \leq r
\end{align*}
\]

where \( R = rs - ks - s \). Again, the intuition behind (3.10) is that only those terms \( b_{i_1, \ldots, i_d} h_1^{i_1} \cdots h_d^{i_d} \), where the sum of the powers of the \( h_i \) is at least \( c \), can contribute to the coefficients in \( f \) of total degree greater than \( cs - s \).
The general multivariate algorithm follows from (3.10) and is analogous to Algorithm 1. In the general algorithm, step (0) requires $O(r^{2d} s^{2d})$ arithmetic operations or $O(r^{2d} s^d d \log(rs))$ operations if $K$ supports a DFT.

In step (2,k) substep (b), we have $O(r^{d-1} s^d)$ linear equations in $O(r^{d-1})$ unknowns. The dimension of the solution space can grow by $O(r^{d-1})$ at each step for $r$ steps, so in the worst case, vector $a_k$ contains linear equations of size $O(r^d)$. To compute the appropriate decomposition of matrix $C_k$ therefore requires $O(r^{3d-3} s^d)$ arithmetic operations, and to solve the resulting system requires an additional $O(r^{5d-2})$ operations. The loop is iterated $r$ times, thus the algorithm requires a total of $O(r^{3d-2} + r^{3d-3} s^d + r^{2d} s^{2d})$ operations, or $O(r^{3d-2} + r^{3d-3} s^d + r^{2d} s^d d \log(rs))$ operations when $K$ supports a DFT.

The dense representation of the input is of size $I = d s^d + n^d$ (the coefficients of the $d$ polynomials $h_1, \ldots, h_d$ and the coefficients of $f$). The output polynomial $g$, in its dense representation, is of size $N = r^d$. In terms of $I$ and $N$, our algorithm requires $O(I^3 N + I^2 N^2)$ operations, or $O(I^3 N + I^2 N \log(IN))$ operations when $K$ supports a DFT.

### 3.2 An Application: Automorphism Inversion in Polynomial Time

We now apply Algorithm 3.1 given in the previous sections to the following two problems:

**Problem 6 (Endomorphism Invertibility)** Given $\sigma \in \text{End}_K K[\bar{x}]$, determine if $\sigma$ is invertible, and if it is, compute its inverse.
Problem 7 (The Inverse of an Automorphism) Given $\sigma \in \text{Aut}_K K[\bar{x}]$, compute its inverse.

### 3.2.1 The Inverse of an Automorphism

Let $h_1(\bar{x}), \ldots, h_d(\bar{x}) \in K[\bar{x}]$. Then $h_1, \ldots, h_d$ uniquely determine an endomorphism $\sigma : K[\bar{x}] \rightarrow K[\bar{x}]$ in which $\sigma(x_i) = h_i(\bar{x})$ and $\sigma(k) = k$ for all $k \in K$. Recall that to be an endomorphism of $K[\bar{x}]$, $\sigma$ must be linear and satisfy $\sigma(gh) = \sigma(g)\sigma(h)$.

By definition, $\sigma$ is an automorphism if $K[\bar{x}] = K[\bar{h}(\bar{x})]$, where $K[\bar{h}(\bar{x})]$ denotes the $K$-subalgebra of $K[\bar{x}]$ generated by $h_1, \ldots, h_d$. It is clear that $K[\bar{h}(\bar{x})] \subseteq K[\bar{x}]$ since $h_i \in K[\bar{x}]$. $K[\bar{x}] \subseteq K[\bar{h}(\bar{x})]$, however, if and only if $x_i \in K[\bar{h}(\bar{x})]$ for $1 \leq i \leq d$. In other words, $K[\bar{x}] \subseteq K[\bar{h}(\bar{x})]$ if and only if there exist polynomials $g_1(\bar{x}), \ldots, g_d(\bar{x}) \in K[\bar{x}]$ such that $g_i(\bar{h}(\bar{x})) = x_i$.

Now let $g_1(\bar{x}), \ldots, g_d(\bar{x}) \in K[\bar{x}]$ be polynomials such that $g_i(\bar{h}(\bar{x})) = x_i$. Let $\psi$ be the endomorphism given by:

$$
\begin{align*}
  x_1 & \mapsto g_1(\bar{x}) \\
  \psi : & \quad : \\
  x_d & \mapsto g_d(\bar{x})
\end{align*}
$$

Since, for $1 \leq i \leq d$, we have $x_i = g_i(\bar{h}(\bar{x}))$, then $x_i \in K[\bar{h}(\bar{x})]$ and thus $K[\bar{x}] = K[\bar{h}(\bar{x})]$ and $\sigma$ is a automorphism with inverse $\psi$.

### 3.2.2 Background and Algorithm

Given an automorphism $\sigma : x_i \mapsto h_i(\bar{x})$, we would like to compute the inverse $\sigma^{-1} : x_i \mapsto g_i(\bar{x})$ of $\sigma$. To do this, we use Algorithm 3.1 for computing the left composition factor in a polynomial decomposition. We will begin with the
bivariate case for which we will give the details and analysis of the algorithm. We will later show how to generalize these results to the multivariate case.

Let \( h_1(x, y), h_2(x, y) \in K[x, y] \) be polynomials with \( \deg h_1 = s_1 \) and \( \deg h_2 = s_2 \). Let \( \sigma \) be an automorphism defined as follows:

\[
\begin{align*}
\sigma : & \quad x \mapsto h_1(x, y) \\
& \quad y \mapsto h_2(x, y)
\end{align*}
\]

(3.12)

Suppose the inverse of \( \sigma \) is given by

\[
\begin{align*}
\sigma^{-1} : & \quad x \mapsto g_1(x, y) \\
& \quad y \mapsto g_2(x, y)
\end{align*}
\]

(3.13)

where

\[
\begin{align*}
g_1(h_1(x, y), h_2(x, y)) &= x \\
g_2(h_1(x, y), h_2(x, y)) &= y
\end{align*}
\]

This leads us to Algorithm 3.2 in Figure 3.3. Given the polynomials \( h_1(x, y) \) and \( h_2(x, y) \), we use Algorithm 3.1 to compute the polynomials \( g_1(z_1, z_2) \) and \( g_2(z_1, z_2) \).

Note that the condition \( \deg h_1 = \deg h_2 \) (i.e. \( s_1 = s_2 \)) stated earlier for Problem 9 is an artificial condition given only to make notation and complexity analysis easier. The algorithm works with only a few minor changes when \( s_1 \neq s_2 \).

It is also interesting to note, however, that even if we kept the condition \( s_1 = s_2 \), we could still use Algorithm 3.2 to compute \( \sigma^{-1} \). Assume that \( s_1 \neq s_2 \). Without loss of generality, let \( s_1 > s_2 \). We set \( h_2'(x, y) = h_1(x, y) + h_2(x, y) \). Now \( \deg h_1 = \deg h_2' \). We run the Algorithm 3.1 as before on inputs \( h_1 \) and \( h_2' \), and compute \( g_1'(x, y) \) and \( g_2'(x, y) \). We now have:

\[
\begin{align*}
g_1'(h_1(x, y), h_2'(x, y)) &= x
\end{align*}
\]
Algorithm 3.2 Automorphism Inversion

Step 1
a) Let $f(x, y) = x$
   b) Let $r = 1$
   c) Run Algorithm 3.1
   d) If “NO DECOMPOSITION”
      then let $r := r + 1$ and go to 1(c)
      else let $g_1(x, y) = g(x, y)$.

Step 2
a) Let $f(x, y) = y$
   b) Let $r = 1$
   c) Run Algorithm 3.1
   d) If “NO DECOMPOSITION”
      then let $r := r + 1$ and go to 2(c)
      else let $g_2(x, y) = g(x, y)$.

Figure 3.3: Algorithm 3.2
\[ g'_2(h_1(x, y), h'_2(x, y)) = y \]

and

\[ h'_2(x, y) = h_2(x, y) + h_1(x, y), \quad (3.14) \]

thus

\[ g'_1(h_1(x, y), h_2(x, y) + h_1(x, y)) = x \]

\[ g'_2(h_1(x, y), h_2(x, y) + h_1(x, y)) = y. \]

We set \( g_1(z_1, z_2) = g'_1(z_1, z_1 + z_2) \) and \( g_2(z_1, z_2) = g'_2(z_1, z_1 + z_2) \). We then have

\[ g_1(h_1(x, y), h_2(x, y)) = g'_1(h_1(x, y), h_2(x, y) + h_1(x, y)) \]

\[ = x \]

and

\[ g_2(h_1(x, y), h_2(x, y)) = g'_2(h_1(x, y), h_2(x, y) + h_1(x, y)) \]

\[ = y. \]

### 3.2.3 Analysis

If \( \sigma \) is an automorphism, then the polynomials \( g_1 \) and \( g_2 \) exist and algorithm 3.2 will terminate. If \( \deg g_1 = r_1 \) and \( \deg g_2 = r_2 \), then the algorithm will be executed \( r_1 \) and \( r_2 \) times, respectively, to compute \( g_1 \) and \( g_2 \). Note that step (0) of Algorithm 3.1 needs to be executed only once. The total number of arithmetic operations required by Algorithm 3.2 to compute \( \sigma^{-1} \) is therefore \( O(r_1^5 + r_1^5 s^2 \log(rs)) \), where \( r = \max(r_1, r_2) \).

In the case of arbitrary dimension \( d \), Algorithm 3.1 is executed \( rd \) times, except for its first step, which is executed only once. This gives us an algorithm
requiring $O(I^3N + I^2N \log(IN))$ algebraic operations to compute the inverse of an automorphism, where $I = ds^d$ is the size of the input in the dense representation and $N = dr^d$ is the size of the output in the dense representation.

### 3.2.4 Endomorphism Invertibility: The Automorphism Question

We now show a final result which follows from the degree bounds on $\sigma^{-1}$ which were shown by Bass, Connell, and Wright [BCW82].

It was shown in [BCW82] that if $K$ is a field and $\sigma$ is an automorphism of $K[\bar{z}]$, then $\deg(\sigma^{-1}) \leq \deg(\sigma)^{d-1}$. In Algorithm 3.2, therefore, the degree $r$ of the output polynomials $g_i$ is at most exponential in $d$ and polynomial in the degree $s$ of the input polynomials $h_i$. In case $K$ is a field, Algorithm 3.2 thus provides us with a solution to Problem 6 that is exponential in the dimension of $K[\bar{z}]$ and polynomial in the degree of $\sigma$.

The solution is as follows: given an endomorphism $\sigma$ that is not known to be an automorphism, we run a version of Algorithm 3.2, but we halt the algorithm at $r = s^d$ if no inverse is found. If no inverse is found, then $\sigma$ is not an automorphism and we return "NO INVERSE." If an inverse is found, we return that inverse.

If the endomorphism $\sigma$ is not an automorphism, we have used an exponential number of arithmetic operations. If it is an automorphism, then we have computed its inverse in time polynomial (sub-cubic) in the dense representation of the input and output. We thus have a polynomial time solution to Problem 7 and an exponential time solution to Problem 6.
3.3 Summary and some Other Results

In this chapter we have presented the first known polynomial time algorithms for two problems: computing the left composition factor of a multivariate decomposition and computing the inverse of an automorphism. To solve Problem 4, computing the left composition factor, we have used techniques similar to those used in the polynomial decomposition algorithms of [KL86]. We then use our solution to Problem 4 in our solution to Problem 7, the automorphism inversion problem. One reason we were able to give a solution to Problem 4 is that we imposed degree constraints on the polynomial \( g \). In Algorithm 3.2, we slowly raise these degree constraints until the inverse is found. The algorithm is guaranteed to terminate, since \( \sigma \) is an automorphism.

One might wonder if this technique would allow polynomial time algorithms based on either a Lagrange inversion formula or on the Gröbner bases techniques of Shannon and Sweedler [SS89]. This is not the case. The Lagrange inversion formulas have an infinite number of terms, and some of these terms can have very high degree but produce terms of low degree only; thus an "algorithm" based on these formulas would not terminate in general. Likewise, the algorithm of Shannon and Sweedler produces a sequence of polynomials on the way to the final output. The polynomials in this sequence can grow exponentially large before shrinking again to produce a small output polynomial. Thus neither of these methods produce polynomial time algorithms, even given degree constraints.

Nonetheless, it is interesting to note that, in cases where a Gröbner basis is already known, then the method of Shannon and Sweedler does produce a polynomial time algorithm. Also, an algorithm based on the inversion formula of
Nousiainen and Sweedler [NS83] can be terminated (Sweedler claims in polynomial time, although no complete analysis has been done) in certain cases when the degree of nilpotence of a set of derivative functions $d_1, \ldots, d_n$ is known. These functions are defined as follows:

\[
\begin{pmatrix}
  d_1 \\
  \vdots \\
  d_n 
\end{pmatrix} = \text{Jac}(\sigma)^{-1} \begin{pmatrix}
  \delta/\delta x_1 \\
  \vdots \\
  \delta/\delta x_n 
\end{pmatrix}
\]  

(3.15)

where the $\text{Jac}(\sigma)^{-1}$ is the inverse of the Jacobian Matrix. It may be that something can be known about the degree of nilpotence of these derivative functions, even when little is known about the degree bounds of the inverse polynomials.
Chapter 4

S-1-Decomposition

In this chapter, we examine another subproblem of Problem 1 which we call the \textit{s-1-decomposition problem}. We will prove that the \textit{s-1-decomposition problem} is NP-hard by reducing the 3-partition problem to it. This implies that the general decomposition problem, Problem 1, is at least NP-hard.

4.1 Background

We begin this section by introducing the \textit{s-1-decomposition problem} as well as a well-known NP-complete problem, the 3-partition problem, which we will later reduce to the \textit{s-1-decomposition problem}. We first define the \textit{s-1-decomposition} and give some of its properties.

4.1.1 Properties of the S-1-Decomposition

\textbf{Definition 4.1} (S-1-Decomposition) \textit{Let} $K$ \textit{be a field and let} $f(x)$ \textit{and} $h(x) \in K[x]$ \textit{be monic univariate polynomials of degree} $n = rs$ \textit{and} $s$, \textit{respectively}. \textit{Let} $g(y, x) \in K[y, x]$ \textit{be a bivariate polynomial of the form}
\[ g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i) \]  

(4.1)

with \( \alpha_i, \beta_i \in \hat{K} \), an algebraic extension of \( K \).

If \( f(x) = g(h(x), x) \) then we call \( g, h \) an \( s \)-1-decomposition of \( f \), and we call \( f \) an \( s \)-1-composition of \( g, h \). \( \Box \)

**Problem 5 (S-1-Decomposition Problem)** Given univariate polynomial \( f(x) \) and an integer \( s \), decide if there exists an \( s \)-1-decomposition \( g, h \) of \( f \). If so, determine the coefficients of \( g \) and \( h \).

We begin with some properties of \( s \)-1-decomposition over the field \( Q \) of rational numbers.

Let \( f(x) \in Q[x] \) be a polynomial of degree \( n = rs, s > 1 \) which factors completely in an extension \( \hat{Q} \) of \( Q \) as

\[ f(x) = \prod_{i=1}^{rs}(x - a_i). \]  

(4.2)

Let \( f \) have an \( s \)-1-decomposition \( g, h \) with

\[ g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i) \]  

(4.3)

for some \( \alpha_i, \beta_i \in \hat{Q} \). We can see that \( f \) and \( h \) are both monic and that the coefficient of \( y^r \) in \( g \) is also 1.

Let \( h(x) + \alpha_i x + \beta_i \) factor in \( \hat{Q} \) as

\[ h(x) + \alpha_i x + \beta_i = \prod_{j=1}^{s}(x - b_{ij}). \]  

(4.4)

Then

\[ f(x) = g(h(x), x) \]
\[= \prod_{i=1}^{r} (h(x) + \alpha_i x + \beta_i)\]
\[= \prod_{i=1}^{r} \prod_{j=1}^{s} (x - b_{ij}) \quad (4.5)\]

From (4.2) and (4.5) we have:
\[f(x) = \prod_{i=1}^{r} (x - a_i) = \prod_{i=1}^{r} \prod_{j=1}^{s} (x - b_{ij}), \quad (4.6)\]
thus \(\{a_i | 1 \leq i \leq n\} = \{b_{ij} | 1 \leq i \leq r, 1 \leq j \leq s\}\).

Two facts follow immediately.

**Fact 4.1** Let \(f(x) \in Q[x]\) be a polynomial which factors completely over \(Q\). If \(f(x)\) has an \(s-1\)-decomposition \(g, h\), then for \(1 \leq i \leq r\), \(h(x) + \alpha_i x + \beta_i\) also factors completely over \(Q\).

If all \(a_i \in Q\), then it follows from Fact 4.1 that all \(\alpha_i\) and \(\beta_i\) are in \(Q\) as well.

**Fact 4.2** Let \(f(x) \in Q[x]\) be a polynomial which factors completely over \(Q\). If \(f\) has any \(s-1\)-decomposition \(g, h\), then there exists an \(s-1\)-decomposition \(g', h'\) such that the constant coefficient of \(h\) and the coefficient of \(x\) in \(h\) are both \(0\). We call this a normal \(s-1\)-decomposition.

To see Fact 4.2, let \(g, h\) be any \(s-1\)-decomposition of \(f\) with \(h(x) = x^s + c_{s-1}x^{s-1} + \cdots + c_2x^2 + c_1x + c_0\). Then the normal \(s-1\)-decomposition \(g', h'\) is given by:
\[g'(y, x) = \prod_{i=1}^{r} (y + (\alpha_i + c_1)x + (\beta_i + c_0)) \quad (4.7)\]
and
\[h'(x) = x^s + c_{s-1}x^{s-1} + \cdots + c_2x^2. \quad (4.8)\]

This give us
\[ f(x) = g(h(x), x) = g'(h'(x), x). \] (4.9)

The following is a generalization of Fact 4.2.

**Fact 4.3** Let \( f(x) \in Q[x] \) be any polynomial which factors over \( Q \) and has an s-1-decomposition \( g, h \) with

\[ g(y, x) = \prod_{i=1}^{r} (y + \alpha_i x + \beta_i). \] (4.10)

Let \( d_0, d_1 \in Q \). If \( h'(x) \in Q[x] \) is some polynomial with \( h'(x) = h(x) + d_1 x + d_0 \), then there exists a polynomial \( g'(x) \) such that \( g', h' \) is an s-1-decomposition of \( f \).

It is easily verified that the polynomial \( g'(y, x) \) is given by:

\[ g'(y, x) = \prod_{i=1}^{r} (y + (\alpha_i - d_1)x + (\beta_i - d_0)). \] (4.11)

### 4.1.2 The 3-Partition problem

Gary and Johnson list the following version of the 3-partition problem in their list of proven NP-hard problems [GJ79].

**Problem 10 (3-Partition Problem (GJ3P))** Given a set \( A \) of \( 3m \) elements, a bound \( B \in Z^+ \), the positive integers, and a size \( \sigma(a) \in Z^+ \) for each \( a \in A \) such that \( B/4 < \sigma(a) < B/2 \) and such that \( \sum_{a \in A} \sigma(a) = mB \), can \( A \) be partitioned into \( m \) disjoint sets \( A_1, \ldots, A_m \) such that, for \( 1 \leq i \leq m, \sum_{a \in A_i} \sigma(a) = B \)? (Note that each \( A_i \) must therefore contain exactly three elements from \( A \).)

In addition, we define the following 3-partition problem Q3P, called the Rational 3-Partition Problem.

**Problem 11 (Rational 3-Partition (Q3P))** Given a set \( A \) of \( 3m \) elements, a bound \( B \in Q \), and a size \( \sigma(a) \in Q \) for each \( a \in A \) such that \( \sum_{a \in A} \sigma(a) = mB \),
can $A$ can be partitioned into $m$ disjoint sets $A_i$, each with 3 elements, such that for $1 \leq i \leq m$, $\sum_{a \in A_i} \sigma(a) = B$?

Any instance of GJ3P is also an instance of Q3P and there is a trivial many-one reduction $\text{GJ3P} \leq_{m}^{\log} \text{Q3P}$, where $\leq_{m}^{\log}$ denotes log-space many-one reducibility [GJ79]. Q3P is therefore NP-hard as well.

### 4.2 Proof of NP-hard

We now prove that the $s$-1-decomposition problem is NP-hard. We will do this by showing that $\text{Q3P} \leq_{m}^{\log} s$-1-decomposition. Our basis for this proof is the following theorem.

**Theorem 4.4** Let $f(x) \in Q[x]$ be a polynomial of degree $n$ which factors completely over $Q$ as $f(x) = \prod_{i=1}^{n}(x - a_i)$. Let $A$ be the set $\{1, 2, \ldots, n\}$ and let $\sigma(i) = a_i, 1 \leq i \leq n$. Then $f(x)$ has an $s$-1-decomposition with $s = 3$ if and only if there is a rational 3-partition of $A$ with size function $\sigma$.

**Proof**

($\Rightarrow$) Let $f(x)$ have an $s$-1-decomposition with $s = 3$. By Fact 4.2, it also has a normal $s$-1-decomposition. Let $g(h(x), x)$ be a normal $s$-1-decomposition of $f(x)$. Let $h(x) = x^3 + cx^2$ and let $g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i)$. Then by Fact 4.1, we can factor $h(x) + \alpha_i x + \beta_i$ as:

$$h(x) + \alpha_i x + \beta_i = \prod_{i=1}^{3}(x - b_{ij}). \tag{4.12}$$

The multisets $\{b_{ij}\}$ and $\{a_i\}$ are equal. Let $B_i = \{b_{i1}, b_{i2}, b_{i3}\}$. Then for $1 \leq i \leq r, r = n/3$, we have:
\[ c = b_{i1} + b_{i2} + b_{i3} \]  \hspace{1cm} (4.13)

and the \( B_i \) are a rational 3-partition of \( A \).

\((\Leftarrow)\) Let \( A = \{a_1, a_2, \ldots, a_n\} \) be the multiset of roots of \( f(x) \) and let \( A \) have a rational 3-partition \( B_1, \ldots, B_r \) with \( B_i = \{b_{i1}, b_{i2}, b_{i3}\} \). Consider \( h \) and \( g \) with
\[
h(x) = x^3 + cx^2 \quad \text{and} \quad g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i),
\]
where \( c = b_{i1} + b_{i2} + b_{i3} \), \( \alpha_i = b_{i1}b_{i2} + b_{i2}b_{i3} + b_{i3}b_{i1} \), and \( \beta_i = b_{i1}b_{i2}b_{i3} \) for \( 1 \leq i \leq r \). Then
\[
h(x) + \alpha_i x + \beta_i = \prod_{j=1}^{3}(x - b_{ij})
\]  \hspace{1cm} (4.14)

and
\[
g(h(x), x) = \prod_{i=1}^{r} \prod_{j=1}^{3}(x - b_{ij}) = \prod_{i=1}^{n}(x - a_i) = f(x).
\]  \hspace{1cm} (4.15)

Thus \( g, h \) gives us an \( s \)-1-decomposition of \( f \). \( \Box \)

Theorem 4.4 provides the basis for the many-one reduction of Q3P to \( s \)-1-decomposition upon which the proof of the following theorem is based.

**Theorem 4.5** The \( s \)-1-decomposition problem is NP-hard.

**Proof** We will prove this theorem by showing the following reduction:

\[
\text{Q3P} \leq_{m}^{\log} \text{s-1-decomposition}.
\]  \hspace{1cm} (4.16)

Let \( A, \sigma, B \) be an instance of Q3P, where \( A \) is a set with \( 3r \) elements, \( \sigma : A \rightarrow Q \), and \( B \in Q \). To reduce Q3P to \( s \)-1-decomposition, we compute the polynomial:
\[
f(x) = \prod_{a \in A} (x - \sigma(a)).
\]  \hspace{1cm} (4.17)
Our instance of the $s$-1-decomposition problem will be the polynomial $f(x)$ with $s = 3$. By Theorem 4.4, there is a 3-partition of $A$ if and only if $f(x)$ has an $s$-1-decomposition. \[\Box\]

We have reduced Q3P to the first part of Problem 5, thus showing that Problem 5 is NP-hard. Moreover, given an $s$-1-decomposition $g, h$ of $f$, we can compute a 3-partition of $A$ in polynomial time, as follows. For $1 \leq i \leq r$, factor $h(x) + \alpha_ix + \beta_i$ as $h(x) + \alpha_ix + \beta_i = \prod_{j=1}^{3}(x - b_{ij})$. Let $B_i = \{b_{i1}, b_{i2}, b_{i3}\}$ for $1 \leq i \leq r$. Then $B_1, \ldots, B_r$ is a rational 3-partition of $A$. It is trivial to compute the polynomial $f(x)$ from the set $A$ in polynomial time. It has also been shown that polynomial factorization over $Q$ can be done in deterministic polynomial time [LLL82].

### 4.3 Conclusion

By showing that Q3P $\leq_{\log}^m s$-1-decomposition, we have shown that Problem 5 is NP-hard. We note that the $s$-1-decomposition problem is an instance of the general decomposition problem, Problem 1. Thus by proving the $s$-1-decomposition problem NP-hard, we have also proven Problem 1, the general polynomial decomposition problem, NP-hard as well.

It is interesting to note that Problem 5, the $s$-1-decomposition problem, is only slightly more general than Problem 2, the univariate decomposition problem. That is, an $s$-1-decomposition $g, h$ of $f$ with $g(y, x) = \prod_{i=1}^{r}(y + \alpha_ix + \beta_i)$ is a univariate decomposition if $\alpha_i = 0$ for $1 \leq i \leq r$. In fact, we can compute the coefficients of $h$ in polynomial time using the univariate decomposition techniques of [KL86]. Beginning with the equation
\[ f(x) = \prod_{i=1}^{r} (h(x) + \alpha_i x + \beta_i), \] 

we note that \( f(x) \) and \( h(x) \)' agree on their coefficients for the highest \( s - 1 \) terms. Following the techniques of [KL86], we can solve for the coefficients of \( h(x) \). However, unlike the univariate decomposition problem, knowing the coefficients of \( h \) does not enable us to solve for the coefficients of \( g \) in the \( s-1 \)-decomposition. This problem remains at least NP-hard.
Chapter 5

Conclusions and Open Problems

In this chapter, we discuss the results given in the previous chapters, and also look ahead toward some work that is still left to be done.

5.1 What Has Gone On Before: A Summary

We have looked at three different decomposition problems, each of which was an instance of the general decomposition problem, Problem 1. We gave polynomial time algorithms for two of these problems and showed the third to be NP-complete.

In Chapter 2, we gave the first known polynomial time algorithm for computing, when it exists, the decomposition of a multivariate polynomial \( f(\vec{x}) \in K[\vec{x}] \) of dimension \( d \) into a univariate polynomial \( g(z) \) and a multivariate polynomial \( h(\vec{x}) \). The algorithm works over any commutative ring \( K \) containing a multiplicative inverse of \( r \), the degree of \( g \), as long as the polynomial \( f \) is monic. For arbitrary dimension \( d \), the algorithm requires \( O(r^{2d+1}s^{3d-1}) \) algebraic oper-
ations, where \( r \) is the degree of \( g \) and \( s \) is the degree of \( h \). We also described an alternative version of that algorithm requiring \( O(r^{d+1}s^{3d}) \) operations. Letting \( N = r^d s^d \) be the size of the input polynomial \( f \) in its dense representation, our algorithms require \( O(N^3) \) arithmetic operations.

We then gave faster versions of the same algorithm requiring only \( O(r^d s^{2d} \log rs) \) operations when \( K \) supports a Fourier transform, and \( O(N^{2d}) \) operations otherwise. Furthermore, using a result from [VSBR81], we showed that the original algorithm could be parallelized so as to run in \( \log^{O(1)} N \) depth using polynomially many processors. We also gave practical parallel speedups of that algorithm using \( N^{1.5} \) processors to yield an \( O(N d) \) depth algorithm, or \( N^3 \) processors to yield an \( O(d \sqrt{N} \log N) \) depth algorithm.

These algorithms represent the first generalization of the univariate decomposition algorithms of [KL86] to the more general case allowing a multivariate \( F \).

In Chapter 3, we gave the first known polynomial time algorithm for the solution to Problem 4, computing the left composition factor of a polynomial decomposition. Our algorithm requires \( O(I^3 N + I^2 N^2) \) arithmetic operations, where \( I \) and \( N \) are the sizes of the input and output polynomials in the dense representation, respectively. When \( K \) supports a Fourier transform, our algorithm requires only \( O(I^3 N + I^2 N \log(IN)) \) arithmetic operations.

The method used in Algorithm 3.1 for computing the left composition factor of a multivariate decomposition is the first such decomposition algorithm to allow the polynomial \( g \) to be multivariate as well as \( f \) and \( h \). Furthermore, this method has an important application to the problem of automorphism inversion.
Algorithm 3.1 provides the means for the first polynomial time solution to the famous and long standing automorphism inversion problem.

Finally, in Chapter 4, we looked at an interesting polynomial decomposition known as $s$-1-decomposition. The related problem is to determine, for a univariate polynomial $f$ and an integer $s$, whether or not $f$ has an $s$-1-decomposition. We proved that the $s$-1-decomposition problem was NP-hard, thus implying that the general decomposition problem, Problem 1, was at least NP-hard.

5.2 What is Left to be Done: A Look Ahead

There are many interesting problems remaining in the area of polynomial decomposition including a number of interesting (or nontrivial) versions of Problem 1. In the first chapter, we defined a nontrivial decomposition for the univariate case, but mentioned that it was not as clear what a trivial or nontrivial solution to the general problem was. In this section, we will discuss this issue and then describe some remaining open problems.

5.2.1 The Interesting, the Difficult, the Trivial, and the Nontrivial

In the general case, we are given a multivariate polynomial $f(\bar{x})$ and we would like to compute some nontrivial functional decomposition $f(\bar{x}) = g(\bar{h}(\bar{x}))$. The first question is: "What do we mean by nontrivial?" We now look at two possible definitions of "nontrivial" as it applies to multivariate polynomial decomposition.

Let $f(\bar{x})$ and the $h_i(\bar{x})$ be $d$-variate polynomials. If $n = d + 1$, then by the following method we can trivially compute functional decompositions $g, h_1, \ldots, h_n$
of $f$ with $g$ an $n$-variate polynomial and the $h_i$'s $d$-variate and of any given degree. Let $f$ be given as follows:

$$f(\bar{x}) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} a_{i_1, \ldots, i_d} x_1^{i_1} \cdots x_d^{i_d}. \quad (5.1)$$

We can choose $h_{n+1}(\bar{x})$ to be any arbitrary polynomial. We then let $h_i = h_{n+1} - x_i$, giving us $x_i = (h_{n+1} - h_i)(\bar{x})$. Now we let

$$g(z_1, \ldots, z_{d+1}) = \sum_{i_1=0}^{n_1} \cdots \sum_{i_d=0}^{n_d} a_{i_1, \ldots, i_d} (z_{n+1} - z_1)^{i_1} \cdots (z_{n+1} - z_n)^{i_d}. \quad (5.2)$$

It is clear that $f(\bar{x}) = g(h(\bar{x}))$. We have computed a decomposition $g, h_1, \ldots, h_n$. Using a similar method, we can always trivially compute a decomposition with an $n$-variate polynomial $g$ if $n$ is greater $d$.

We would like to define a nontrivial decomposition in such a way as to exclude such trivial cases, while still including the more interesting and difficult problems. Given a $d$-variate polynomial $f$, we can see from (5.2) that most of the interesting decomposition problems seem to be those in $g$ is $n$-variate with $n \leq d$.

In our discussion of endomorphisms in Chapter 3, $f$ and $g$ were both $d$-variate polynomials. That is, $n = d$. For every indeterminate $x_i$, we had one polynomial $h_i(\bar{x})$ specifying $x_i$'s image. Following this idea, let us assume that $g$ is also a $d$-variate polynomial. To see another type of trivial decomposition, let $\sigma$ be an automorphism of $K[\bar{x}]$ with $\sigma$ given by $\sigma : x_i \mapsto h_i(\bar{x})$. Let $\psi$ be the inverse of $\sigma$ with $\psi$ given by $\psi : x_i \mapsto h_i'(\bar{x})$. Then

$$f(\bar{x}) = \psi \sigma f(\bar{x})$$

$$= \psi f(h(\bar{x})). \quad (5.3)$$

Thus for a given automorphism $\sigma$ there is always a decomposition $g, h_1, \ldots, h_d$ of $f$ where $g(\bar{x}) = \psi f(\bar{x}) = f(h'(\bar{x}))$. One might ask whether or not this decom-
position was really trivial, since given an endomorphism \( \sigma : x_i \mapsto h_i(\overline{x}) \), it still not known how to determine in polynomial time whether this is an automorphism. However, it is fairly simple to produce an automorphism of any degree (see [Nag72]), and we can compute the inverse of this automorphism either as we produce it, or in polynomial time using the method given in Chapter 3.

We now have our first and most intuitive definition of a nontrivial multivariate functional decomposition which excludes the above trivial cases.

**Definition 5.1 (Nontrivial Decomposition)** A decomposition \( g, h_1, \ldots, h_n \) of a \( d \)-variate polynomial \( f(\overline{x}) \) is called nontrivial if \( n = d \) and if the endomorphism \( \sigma : x_i \mapsto h_i \) for \( 1 \leq i \leq d \) is not an automorphism. \( \square \)

A second notion of nontrivial follows from the idea of a *lossless* decomposition. This also excludes many of the trivial decompositions computed from automorphisms.

**Definition 5.2 (Lossless Decomposition)** A decomposition \( g, h_1, \ldots, h_n \) of the \( d \)-variate polynomial \( f \) is called lossless if, for \( 1 \leq i \leq d \):

\[
n_i = \sum_{j=1}^{n} r_j s_{i,j}
\]

(5.4)

where \( n_i = \deg_{x_i} f, r_j = \deg_{z_j} g, \) and \( s_{i,j} = \deg_{z_i} h_j \). \( \square \)

From this definition of a lossless decomposition, we get the following definition of a nontrivial decomposition.

**Definition 5.3 (Nontrivial Lossless Decomposition)** A decomposition \( g, h_1, \ldots, h_n \) of the \( d \)-variate polynomial \( f \) is called a nontrivial lossless decomposition if it is lossless and if \( n = d \), \( \deg g > 1 \) and \( \deg h_i > 1 \) for \( 1 \leq i \leq d \).
Let $f$ have a nontrivial decomposition $g, h_1, \ldots, h_d$ (using either definition of "nontrivial.") We may now ask if $h_1, \ldots, h_d$ also have nontrivial decompositions. A more difficult question, and one which may be interesting in light of our discussion of endomorphisms and left composition factors, is whether $h_1, \ldots, h_d$ have nontrivial decompositions that share common right composition factors. That is, is there some set of polynomials $h'_1, \ldots, h'_d$ (or equivalently some endomorphisms $\sigma$ mapping $x_i \mapsto h'_i(\vec{x})$) such that there exist left composition factors $h_{1,1}, \ldots, h_{1,d}$ composing with the $h'_1, \ldots, h'_d$ to give $h_1, \ldots, h_d$? If so, then we have:

$$f(\vec{x}) = g \circ (h_1, \ldots, h_d)$$
$$= g \circ (h_{1,1}, \ldots, h_{1,d}) \circ (h'_1, \ldots, h'_d).$$

Likewise, if $h'_1, \ldots, h'_d$ all decompose with common right composition factors $h_{3,1}, \ldots, h_{3,d}$ and left composition factors $h_{2,1}, \ldots, h_{2,d}$, then we have:

$$f(\vec{x}) = g \circ (h_{1,1}, \ldots, h_{1,d}) \circ (h_{2,1}, \ldots, h_{2,d}) \circ (h_{3,1}, \ldots, h_{3,d})$$

This leads us to our notion of a fully extended decomposition.

**Definition 5.4 (Fully Extended Decomposition)** A nontrivial decomposition

$$f(\vec{x}) = g \circ (h_{1,1}, \ldots, h_{1,d}) \circ \cdots \circ (h_{e,1}, \ldots, h_{e,d})$$

is called a fully extended decomposition of $f(\vec{x})$ if there do not exist nontrivial decompositions

$$h_{i,1}(\vec{x}) = g_1(h_1(\vec{x}), \ldots, h_d(\vec{x}))$$
$$\vdots$$
$$h_{i,d}(\vec{x}) = g_d(h_1(\vec{x}), \ldots, h_d(\vec{x}))$$

for any $g_1, \ldots, g_d$ and $h_1, \ldots, h_d \in K[\vec{x}]$, and $1 \leq i \leq e$. □
5.2.2 Open Problems

The first two open problems follow from Definition 5.1 or 5.3 of a nontrivial decomposition and from Definition 5.4.

**Problem 12 (Nontrivial Multivariate Decomposition)** Given multivariate polynomial \( f(\vec{x}) \in K[\vec{x}] \), determine if \( f \) has a nontrivial functional decomposition \( f(\vec{x}) = g(\vec{h}(\vec{x})) \). If so, compute the coefficients of \( g \) and the \( h_i \).

**Problem 13 (Fully Extended Decomposition)** Given a multivariate polynomial \( f(\vec{x}) \in K[\vec{x}] \), compute a fully extended decomposition.

The following problem is another instance of Problem 1.

**Problem 14 (Multivariate Decomposition of Given Degree)** Given polynomial \( f(\vec{x}) \in K[\vec{x}] \) and some subset of the following: integers \( n, r, s_1, \ldots, s_n \), polynomial \( g(\vec{x}) \in K[\vec{x}] \), and polynomials \( h_1(\vec{x}), \ldots, h_n(\vec{x}) \in K[\vec{x}] \), decide if there exists a functional decomposition \( g, h_1, \ldots, h_n \) of \( f \) such that \( \deg g = r \) and \( \deg h_i = s_i \) for \( 1 \leq i \leq n \). If so, compute those coefficients of \( g \) and the \( h_i \)'s which were not given.

Problem 14 is only slightly less general than Problem 1. Problem 1 allows, as part of the input, a template specifying the form of the output polynomials, whereas Problem 14 allows only the degree of the output polynomials to be specified as input. However, whereas Problem 5 is an instance of Problem 1, it is not an instance of Problem 14. This leads to another question.

**Problem 15 (Open)** Is Problem 14 NP-hard?
The $s$-1-decomposition problem seems intuitively easier than Problem 14. In Problem 14, $f, g$ and $h$ are general multivariate polynomials of arbitrary dimension. Furthermore, polynomial $g$ takes the polynomial $h_i$ as arguments, and we know nothing about the form of $g$ other than its degree. In the $s$-1-decomposition problem, on the other hand, $f$ and $h$ are both univariate polynomials and $g$ is only bivariate. Furthermore, $g$ takes $x$ and not another polynomial as its second argument. We also know a great deal about the structure of the polynomial $g$, namely that it factors as:

$$g(y, x) = \prod_{i=1}^{r}(y + \alpha_i x + \beta_i).$$

However, we have tried without success to reduce the $s$-1-decomposition problem to Problem 14.

**Problem 16 (Open)** Is the $s$-1-decomposition problem in NP over the field $\mathbb{Q}$ of rational numbers?

A final set of problems deals with the representation of the polynomials. Andrew Quirk [Qui86] lists three representations of polynomials common in the field of computational algebra. These are: the dense representation, which lists all coefficients in some order; the sparse representation, which lists only the nonzero coefficients and their corresponding exponents; and the straight-line program, a branch-free program which describes the polynomials in terms of elementary operations on basis polynomials. The sparse representation is clearly more efficient than the dense representation for polynomials with proportionally many zero coefficients. Furthermore, algorithms which are polynomial time in the dense representation are not guaranteed to be polynomial or even exponential.
in the sparse representation. There has been much recent interest in efficient algorithms for factoring sparse polynomials [vzG83,vzGK85], in algorithms for sparse polynomial interpolation [BOT88,KY88,Zip89], and in GCD and divisibility problems for sparse polynomials [Qui86]. Algorithms 2.1, 2.2, 3.1 and 3.2 all assume a dense representation of the input polynomials, as do the univariate decomposition algorithms of Kozen and Landau [KL86,KL89] and the univariate and multivariate decomposition algorithms of von zur Gathen [vzG87,vzG88].

**Problem 17 (Open)** Are there fast (polynomial time) algorithms for solutions to Problems 2, 3, 4 and 6 when the input polynomials are sparse?
Bibliography


