Computing Polynomial Composition Factors and Inverses of Automorphisms in Polynomial Time*

Matthew Dickerson

88-955
December 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported by NSF grant CCR88- 06979.

Computing Polynomial Composition Factors and Inverses of Automorphisms in Polynomial Time *

Matthew Dickerson

November 5, 1988

Abstract

The problem of determining when an endomorphism on a polynomial ring is an automorphism and, when it is, the problem of computing its inverse are long standing problems which have received much attention both recently and in the past [BCW], [N], [NS], [SS]. There has also been much attention given recently to various problems relating to the functional decomposition of polynomials [AT],[BZ],[D],[G1],[G2],[GKL],[KL].

In this paper, we present the first polynomial time algorithm for computing the left composition factor of a multivariate polynomial decomposition and we use that result in the first polynomial time algorithm for computing the inverse of an automorphism on a polynomial ring. Finally, we show how these algorithms can be used to determine when an endomorphism is an automorphism.

*Supported by NSF grant CCR88-06979

1 Introduction

Problem 1 (Inverse of an Automorphism) *Given an automorphism σ in $K[\vec{x}] \rightarrow K[\vec{x}]$, compute its inverse.*

Problem 2 (Left Composition Factor of a Polynomial Decomposition) *Let polynomials $f(\vec{x}), h_1(\vec{x}), \ldots, h_d(\vec{x}) \in K[\vec{x}]$ and integer r be given. Does there exist a polynomial $g(\vec{z}) \in K[\vec{z}]$ of total degree less than or equal to r which composes the h's to give f? (That is, does there exist a polynomial $g(\vec{z})$ such that $f(\vec{x}) = g(\vec{h}(\vec{x}))$ and $\deg g \leq r$?) If so, compute one.*

A long standing problem in computational algebra has been to compute the inverse of an autorphism $\sigma \in K[\vec{x}] \rightarrow K[\vec{x}]$. A previous algorithm, due to Shannon and Sweedler [SS], requires the computation of a Gröbner basis and is exponential time. There are also many known inversion formulas for σ^{-1} ([BCW],[NS]) but all of them have an infinite number of terms and thus an algorithm based on those formulas would be non-terminating.

A second area which has received much attention in recent years is that of polynomial decomposition. In 1986, Kozen and Landau [KL] gave the first polynomial time algorithm for obtaining a non trivial functional decomposition $f(x) = g(h(x))$ of a univariate polynomial over a commutative ring K. Previous algorithms ([AT],[BZ]) were exponential time, required polynomial factorization, and only worked over fields of characteristic 0. In 1987, Dickerson [D] gave the first polynomial time algorithm for the more general functional decomposition of multivariate f into a univariate g and a multivariate h. The algorithms of both [KL] and [D] work over any commutative ring K in the "tame case" when the ring contains a multiplicative inverse of r, the degree of h. Von zur Gathen [G1], [G2] has since given faster algorithms for both the univariate and multivariate decompositions in the "tame case" as well as some partial results for the "wild case" when the characteristic of K divides the degree of h. The problem of the general multivariate decomposition of $f(\vec{x})$ into $g(\vec{h}(\vec{x}))$ in polynomial time is still open.

In this paper we extend some techniques used in polyonmial decomposition to present the first polynomial time algorithm for computing the left composition factor in a general multivariate composition (problem 2) and we use this result in the first polynomial time algorithm for computing the inverse of an automorphism (problem 1).

1.1 History

Problem 1 is a subproblem of the following problem.

Problem 3 (Endomorphism Invertibility) *Given an endomorphism $\sigma : x_i \rightarrow h_i(\vec{x}) \in K[\vec{x}]$, determine if σ is invertible, and if so, compute its inverse.*

Restated, problem 3 is to determine when an endomorphism is an automorphism, and when it is, to compute its inverse. This problem has received much attention and there is a vast amount of

literature on the subject of characterizing the group of automorphisms ([BCW],[N],[NS],[SS]). Two famous conjectures are:

Conjecture 1 *$Aut_K K[\vec{x}]$ is generated by $GL(n,K)$, $J(n,K)$, and $N(n,K)$ where GL, J, and N are, respectively, the general linear group, the Jonquière automorphisms, and the subgroup of nilpotency.*

Conjecture 2 (Jacobian Conjecture) *If K contains the field Q of rational numbers and σ is an endomorphism of $K[\vec{x}]$ with the Jacobian matrix of σ invertible, then σ is an automorphism of $K[\vec{x}]$.*

Both conjectures are true under certian conditions but false in the general case when dimension d of $\vec{x}$ is arbitrary and when the field K has arbitrary characteristic. See [N] and [BCW] for good summaries of these conjectures and related work on the automorphism group.

In this paper, rather than presenting a mathematical characterization of automorphisms, we present a fast (subcubic on the size of the input and output) algorithm for actually computing that inverse when it exists. Much work has been done on the problem of computing the inverse of σ in the case when σ is an automorphism. There are a number of inversion formulas which give σ^{-1} in terms of the inverse of the Jacobian matrix. Lagrange Inversion Formulas such as that of Abhyankar give a nice mathematical model for the inverse but it and other similar formulas such as that in [NS] may not terminate. Shannon and Sweedler [SS] recently gave an elegant algorithm to compute σ^{-1} which uses Gröbner bases techniques, but the computation of a Gröbner basis requires exponential time.

The algorithm presented here for computing the left composition factor in a multivariate polynomial decomposition requires a number of arithmetic operations which is subcubic on the size of the input and output polynomials in the dense representation. It can be used to compute the inverse of an automorphism in subcubic time. These are the first known polynomial time algorithms for the solutions of these problems.

1.2 Notation

Throughout this paper, let K be a commutative ring with $K[x_1,\ldots,x_d]$ being the d-variate polynomial ring over K. We use the standard notation $K[\vec{x}]$ for $K[x_1,\ldots,x_d]$.

By $\deg f$ we shall mean the total degree of f.

By $f_1 f_2 \cdots f_n(\vec{x})$, we shall mean the product $f_1(\vec{x})f_2(\vec{x})\cdots f_n(\vec{x})$. By $f_1^{i_1} f_2^{i_2} \cdots f_n^{i_n}(\vec{x})$ we shall mean the product $f_1(\vec{x})^{i_1} f_2(\vec{x})^{i_2} \cdots f_n(\vec{x})^{i_n}$.

If $g(\vec{z})$ and $h_1(\vec{x}),\ldots,h_d(\vec{x})$ are polynomials, then by $g(\vec{h}(\vec{x}))$, we shall mean the functional composition $g(h_1(\vec{x}),\ldots,h_d(\vec{x}))$ of g and the h_i.

Finally, by $g(\vec{x}) \equiv_{(>i)} f(\vec{x})$ we shall mean that $g(\vec{x})$ and $f(\vec{x})$ have equal coefficient for all terms of degree greater than i where i is some integer exression.

1.3 Background

Let $h_1(\vec{x}), \ldots, h_d(\vec{x}) \in K[\vec{x}]$ be polynomials and let $\sigma : K[\vec{x}] \rightarrow K[\vec{x}]$ be the endomorphism defined by:

$$\sigma : \begin{array}{rcl} x_1 & \rightarrow & h_1(\vec{x}) \\ & \vdots & \\ x_d & \rightarrow & h_d(\vec{x}) \end{array} \tag{1}$$

σ is an automorphism if and only if it has an inverse. That inverse is defined by the polynomials $g_1(\vec{x}), \ldots, g_d(\vec{x}) \in K[\vec{x}]$ such that $g_i(\vec{h}(\vec{x})) = x_i$. We can see that if this set of polynomials g_i exists for $1 \leq i \leq d$ then $K[\vec{x}] = K[\vec{h}(\vec{x})]$. Furthermore, it is easily verified that $\psi = \sigma^{-1}$ is given by:

$$\psi : \begin{array}{rcl} x_1 & \rightarrow & g_1(\vec{x}) \\ & \vdots & \\ x_d & \rightarrow & g_d(\vec{x}) \end{array} \tag{2}$$

$$\begin{aligned} \sigma\psi : x_i &= \sigma g_i(\vec{x}) \\ &= g_i(\sigma x_1, \ldots, \sigma x_d) \\ &= g_i(\vec{h}(\vec{x})) \\ &= x_i \end{aligned}$$

The problem is to compute the polynomials g_i. To do this, we use 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.

2 Bivariate Decomposition: Computing the Left Factor

In this section, we present an algorithm for solving the following problem:

Problem 4 *Let polynomials $f(x,y), h_1(x,y)$ and $h_2(x,y) \in K[x,y]$ and integer r be given. Let $n = \deg f$ and let $s = \deg h_1 = \deg h_2$. Does there exist 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 one.*

2.1 Basis for the Algorithm

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

$$g(z_1, z_2) = \sum_{i=0}^{r} \sum_{j=0}^{r-i} b_{i,j} z_1^i z_2^j \tag{3}$$

It is easy to show that

$$f(x, y) \equiv_{(>rs-s)} \sum_{i=0}^{r} b_{i,r-i} h_1^i h_2^{r-i}(x, y) \tag{4}$$

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:

$$f(x, y) \equiv_{(>rs-s-ks)} \sum_{i=0}^{r} \sum_{j=r-i-k}^{r-i} b_{i,j} h_1^i h_2^j(x, y) \tag{5}$$

and thus $\deg(f(x, y) - \sum_{i=0}^{r} \sum_{j=r-i-k}^{r-i} b_{i,j} h_1^i h_2^j(x, y)) \leq rs - s - ks$. From simple substitution we can now see that

$$f(x, y) - \sum_{i=0}^{r} \sum_{j=r-i-(k-1)}^{r-i} b_{i,j} h_1^i h_2^j(x, y) \equiv_{(>rs-s-ks)} \sum_{i=0}^{r} b_{i,r-i-k} h_1^i h_2^{r-i-k}(x, y) \tag{6}$$

This tells us that only products $h^i(x, y)h^j(x, y)$ where $i + j > k$ can produce terms of degree $> ks$ thus only those coefficents $b_{i,j}$ of g where $i + j > k$ affect the coefficients of f of degree $> ks$. If we can compute the coefficients $b_{i,j}$ for $i + j = r$ we can use these coefficients to compute the coefficients $b_{i,j}$ for $i + j = r - 1, r - 2$,etc. We can compute coefficients $b_{i,r-i-k}$ for $0 \leq i \leq r - k$ in terms of $b_{i,r-i-j}$ for $0 \leq i \leq r - j$ and $j < k$.

This is exactly what we do in Algorithm 1 given in Figure 1.

2.2 More On Step $2.k$ (b)

Algorithm 1 hinges on solving the system of equations in step 2 part (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 $r + 1 - k$ unknowns: $b_{0,r-k}, b_{1,r-k-1}, \ldots, b_{r-k,0}$. Let us define C_k, b_k, and a_k as in Figure 2, then this system is given by the equation:

$$C_k b_k = a_k \tag{7}$$

The system given by Equation 7 is overdetermined but may or may not be rank deficient. In step 2 part (b) we may get no solution, a unique solution, or a solution space of dimension $\leq r - k$.

It is clear from Equation 6 that if there is no solution to Equation 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 with no anomalies.

Step 0) For $0 \leq i \leq r$
For $0 \leq j \leq r - i$
Compute $h_1^i h_2^j(x, y)$.
Let $c_{i,j,k,l}$ be the coefficient of $x^k y^l$ in $h_1^i h_2^j(x, y)$.
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 2)
c) Let $f_{k+1}(x, y) := f_k(x, y) - \sum_{l=0}^{r-k} b_{l,r-l-k} h_1^l h_2^{r-l-k}(x, y)$.
Step 3) Let $b_{0,0} := f_r(x, y)$.

□

Figure 1: Algorithm 1

If at step $2.k$ part (b) we get a solution space rather than a unique solution, then in part (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 following step, step $2.(k + 1)$, we must solve for $b_{0,r-(k+1)}, b_{1,r-(k+1)-1}, \cdots, b_{r-(k+1),0}$ 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 chose some $r - k$ rows which are linearly independent, call these rows C'_{k+1} and 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 unique 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 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 it will increase by at most $r - k - 1$. The important thing to note is that in the case when we have a solution space rather than a unique solution, 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) 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.

$$C_k = \begin{bmatrix} c_{0,r-k,0,r-ks-s+1} & c_{1,r-k-1,0,rs-ks-s+1} & \cdots & c_{r-k,0,0,rs-ks-s+1} \\ c_{0,r-k,0,r-ks-s+2} & c_{1,r-k-1,0,rs-ks-s+2} & \cdots & c_{r-k,0,0,rs-ks-s+2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{0,r-k,0,rs-ks} & c_{1,r-k-1,0,rs-ks} & \cdots & c_{r-k,0,0,rs-ks} \\ c_{0,r-k,1,r-ks-s} & c_{1,r-k-1,1,rs-ks-s} & \cdots & c_{r-k,0,1,rs-ks-s} \\ c_{0,r-k,1,r-ks-s+1} & c_{1,r-k-1,1,rs-ks-s+1} & \cdots & c_{r-k,0,1,rs-ks-s+1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{0,r-k,rs-ks,0} & c_{1,r-k-1,rs-ks,0} & \cdots & c_{r-k,0,rs-ks,0} \end{bmatrix}$$

$$b_k = \begin{bmatrix} b_{0,r-k} \\ b_{1,r-k-1} \\ b_{2,r-k-2} \\ \vdots \\ b_{r-k,0} \end{bmatrix}$$

$$a_k = \begin{bmatrix} a_{k,0,rs-ks-s+1} \\ a_{k,0,rs-ks-s+2} \\ \vdots \\ a_{k,0,rs-ks} \\ a_{k,1,rs-ks-s} \\ a_{k,1,rs-ks-s+1} \\ \vdots \\ a_{k,rs-ks,0} \end{bmatrix}$$

Figure 2: $C_k b_k = a_k$

2.3 Analysis

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

$$h_1^i h_2^j(x,y) := h_2(x,y)(h_1^i h_2^{j-1}(x,y)) \tag{8}$$

This later method requires only $O(r^4 s^4)$ 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 multiplications, and then doing an inverse transform on the resulting products. The multivariate DFT requires $O(n^2 \log n)$ time for a bivariate polynomial of degree n. (See [AFW] or [NQ]). Step 0 therefore requires $O(r^4 s^2 \log(rs))$ arithmetic operations.

Step 2 The system of equations in step 2 part (b) has size $[(r-k+1/2)s^2 + s/2] \times [r+1-k]$ which is $O(rs^2) \times O(r)$. The $a_{k,i,j}$ are given by linear equations of length 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, for instance, the techniques of [GL] applied symbolically. r iterations of step 2 therefore require $O(r^5 + r^4 s^2)$ arithmetic operations.

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 on the size of the input.

3 The Inverse of an Automorphism

Algorithm 1 can be applied to the problem of computing the inverse of an automorphism.

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 σ be an automorphism defined as follows:

$$\sigma : \begin{array}{ccc} x & \rightarrow & h_1(x,y) \\ y & \rightarrow & h_2(x,y) \end{array} \tag{9}$$

σ^{-1}, the inverse of σ, is defined by

$$\sigma^{-1} : \begin{array}{ccc} x & \rightarrow & g_1(x,y) \\ y & \rightarrow & g_2(x,y) \end{array} \tag{10}$$

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

□

Figure 3: Algorithm 2

where

$$\begin{aligned} g_1(h_1(x, y), h_2(x, y)) &= x \\ g_2(h_1(x, y), h_2(x, y)) &= y \end{aligned}$$

This leads us to Algorithm 2 in Figure 3. Given the polynomials $h_1(x, y)$ and $h_2(x, y)$ we use Algorithm 1 to compute the polynomials $g_1(z_1, z_2)$ and $g_2(z_1, z_2)$.

Note that the condition $\deg h_1 = \deg h_2$, (ie. $s_1 = s_2$) stated earlier for Problem 4 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 interesting to note, however, that even if we kept the condition $s_1 = s_2$ we could still use Algorithm 1 to compute σ^{-1}. Assume that $s_1 > s_2$ then 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 2 as before except with inputs h_1 and h_2' and compute $g_1'(x, y)$ and $g_2'(x, y)$. We now have:

$$\begin{aligned} g_1'(h_1(x, y), h_2'(x, y)) &= x \\ g_2'(h_1(x, y), h_2'(x, y)) &= y \end{aligned}$$

and

$$h_2'(x, y) = h_2(x, y) + h_1(x, y) \tag{11}$$

thus

$$\begin{aligned} g_1'(h_1(x,y), h_2(x,y)+h_1(x,y)) &= x \\ g_2'(h_1(x,y), h_2(x,y)+h_1(x,y)) &= y \end{aligned}$$

So we set $g_1(z_1, z_2) = g_1'(z_1, z_1+z_2)$ and likewise we set $g_2(z_1, z_2) = g_2'(z_1, z_1+z_2)$ and we are done.

$$\begin{aligned} g_1(h_1(x,y), h_2(x,y)) &= x \\ g_2(h_1(x,y), h_2(x,y)) &= y \end{aligned}$$

3.1 Analysis

If σ is an automorphism, then the polynomials g_1 and g_2 exist. Let $\deg g_1 = r_1$ 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 1 needs to be executed only once. Total number of arithmetic operations required by Algorithm 2 to compute σ^{-1} is therefore $O(r^6 + r^5 s^2 \log(rs))$ where $r = \max(r_1, r_2)$.

4 Multivariate Decomposition: The General Case

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

For ease of notation, we first define a new function σ_d^j. Let I be the set $I = \{i_1, i_2, \ldots, i_d\}$ and let σ_d^j be the function on I defined as follows:

$$\sigma_d^j(I) = \sum_{k=1}^{j} i_k \tag{12}$$

Also let

$$g(\vec{z}) = \sum_{i_1=0}^{r} \sum_{i_2=0}^{r-i_1} \cdots \sum_{i_d=0}^{r-\sigma_d^{d-1}(I)} b_{i_1,i_2,\ldots,i_d} z_1^{i_1} z_2^{i_2} \cdots z_d^{i_d} \tag{13}$$

We now give with the multivariate analog to Equation 5 which is fairly easy to prove.

$$f(\vec{x}) \equiv_{(>rs-ks-s)} \sum_{i_1=0}^{r} \sum_{i_2=0}^{r-i_1} \cdots \sum_{i_{d-1}=0}^{r-\sigma_d^{d-2}(I)} \sum_{i_d=r-\sigma_d^{d-1}(I)-k}^{r-\sigma_d^{d-1}(I)} b_{i_1,i_2,\ldots,i_d} h_1^{i_1} h_2^{i_2} \cdots h_d^{i_d}(\vec{x}) \tag{14}$$

The general multivariate algorithm follows from Equation 14 and is analgous 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 using a DFT. In step 2.k part (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. Step 2 therefore requires a total of $O(r^{3d-2} + r^{3d-3}s^d)$ arithmetic operations. The dense representation of the input is of size $o(r^d)$ and the output of size $o(s^d)$ so we still have a polynomial time algorithm in dense representations of the input and output.

5 Conclusion and Open Problems

In this paper we have presented the first polynomial time algorithms for two significant problems. To solve Problem 2 we have used some techniques similar to those used in the polynomial decomposition algorithms of [KL]. We then use our solution to Problem 2 in our solution to Problem 1, the automorphism inverse problem. One reason we were able to give a solution to Problem 2 is that we imposed degree constraints on the polynomial g. In Algorithm 2 we slowly raise these degree constraints until the inverse is found. The algorithm is guaranteed to terminate since σ is an automorphism.

One might wonder if this technique would allow polynomial time algorithms based on either a Lagrange inversion formulas or on the Gröbner bases techniques of [SS]. This is not the case. The Lagrance 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. 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.

A final result follows from degree bounds on σ^{-1} given by [BCW]. It was shown in [BCW] that if K is a field and σ is an automorphism in $K[\vec{x}]$ then $\deg(\sigma^{-1}) \leq \deg(\sigma)^{d-1}$. The degree r of the output polynomials g_i in Algorithm 2 are therefore at most exponential size in the dimension d and polynomial size in the degree s of the input polynomials h_i. In the case when K is a field Algorithm 2 provides us with a solution to Problem 3 which is exponential in the dimension of $K[\vec{x}]$ and polynomial in the degree of σ. Given an endomorphism σ which is not known to be an automorphism, we run a version of Algorithm 2 but we can halt the algorithm at $r = s^d$ if no inverse is found. If endomorphism σ is not an automorphism, we have used an exponential number of arithmetic computations. If it is an automorphism, then we have computed its inverse in time polynomial (sub-cubic) on the dense representation of the input and output. We thus have a polynomial time solution to Problem 1 and an exponential time solution to Problem 3.

5.1 Open Problems

The algorithms given in this paper assumes a dense representation of the input and output polynomials

Problem 5 (Open) *Is there an asymptotically fast algorithm for sparse polynomials?*

Problem 6 (Open) *Is there an asymptotically fast algorithm to compute the right composition factors $h_1, \ldots, h_d$ in a decomposition $f(\vec{x}) = g(\vec{h}(\vec{x}))$.*

Our solution to Problem 3, the Endomorphism Invertibility Problem, is dependent on their being a bound on the degree of the inverse of σ. A tighter bound in special cases might give a polynomial time algorithm for the solution of this problem.

Problem 7 (Open) *The degree bound on σ^{-1} given in [BCW] holds when K is a field. Is there a bound on* $\deg(\sigma^{-1})$ *when K is a commutative ring but not a field? Is there a tighter bound on* $\deg(\sigma^{-1})$ *for any special cases of K?*

References

[AFW] *Auslander, L., E. Feig and S. Winograd. "Algorithms for the Multidimensional Discrete Fourier Transform".* IEEE Trans. on Acoust. Speech, and Signal Processing *April 1983, 388-403*

[AT] *Alagar, V.S. and M. Thanh. "Fast Polynomial Decomposition Algorithms".* Proc. EUROCAL85, Lect. Notes in Comp. Sci. 204, *Springer-Verlag, Heidelberg, 1985, 150-153.*

[BZ] *Barton, D.R. and R.E. Zippel. "Polynomial Decomposition Algorithms".* Journal Symb. Comp *(1985), 159-168.*

[BCW] *Bass, H., E. Connell, and D. Wright. "The Jacobian Conjecture: Reduction of Degree and Formal Expansion of the Inverse".* Bull. of the Amer. Math. Soc. *Vol 7, No 2; September 1982, 287-330.*

[D] *Dickerson, M. "Polynomial Decomposition Algorithms for Multivariate Polynomials". Tech. Rep. TR87-826, Dept. of Comp. Sci., Cornell Univ., April 1987*

[G1] *von zur Gathen, J. "Functional Decomposition of Polynomials: the Tame Case".* manuscript *Sept. 1987*

[G2] *von zur Gathen, J. "Functional Decomposition of Polynomials: the Wild Case".* manuscript *June 1988*

[GKL] *von zur Gathen, J., D.Kozen, and S. Landau. "Functional Decomposition of Polynomials".* Proc. 28th IEEE Symp. Found. Comp. Sci., *Nov. 1987, 127-131.*

[GL] *Golub, van Loan "Matrix Computations". Johns Hopkins, 1983.*

[KL] *Kozen, D. and S. Landau. "Polynomial Decomposition Algorithms". Tech. Rep. TR86-773, Dept. of Comp. Sci., Cornell Univ.* Journal Symb. Comp. *to appear.*

[N] *Nagata, M. "On Automorphism Group of $k[x, y]$". Department of Mathematics, Kyoto University.* Kinokuniya Book-store co., Ltd. Tokyo, Japan *1972*

[NQ] *Nussbaumer, H. and P. Quandalle. "Fast Computation of Discrete Fourier Transforms Using Polynomial Transforms"* . IEEE Trans. on Acoust. Speech, and Signal Processing *April 1979, 169-181*

[NS] *Nousiainen, P. and M. Sweedler. "Automorphisms of Polynomial and Power Series Rings"*. Journal of Pure and Applied Algebra *29 1983, 93-97*

[R] *Ritt, J.F. "Prime and Composite Polynomials"*. Transactions of Amer. Math. Soc. *23 (1922), 51-66*

[SS] *Shannon, D. and M. Sweedler. "Using Gröbner Bases to Determine Algebra Membership, Split Surjective Algebra Homomorphisms and Determine Birational Equivalence"*. JACM *To appear.*