A Comparison of Two Families of Algorithms for Symbolic Polynomials

(Spline Title: A Comparison of Algorithms for Symbolic Polynomials)
(Thesis Format: Monograph)

by

Matt Malenfant

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
The University of Western Ontario
London, Ontario, Canada
December, 2007

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

Chief Adviser:

Dr. Stephen Watt

Advisory Committee:

Examining Board:

Dr. Dan Christensen

Dr. Marc Moreno Maza

Dr. Eric Schost

The thesis by
Matt Malenfant

entitled:
A Comparison of Two Families of Algorithms for Symbolic Polynomials

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date: _____

Chair of Examining Board
Dr. Charles Ling

Abstract

Symbolic polynomials, whose exponents themselves are integer-valued multivariate polynomials, arise often in algorithm analysis. Unfortunately, modern computer algebra systems do not provide ample support for their algebraic structures. Basic operations involving symbolic polynomials are indeed trivial (addition, multiplication, derivatives); however, other crucial operations remain much more difficult, such as factorization and GCD.

Watt has given two separate methods to solve these challenging problems. The first uses a change of variables, e.g. x^n to X, x^{nm} to Y. Doing so increases the number of variables potentially exponentially. Evaluation/Interpolation is used in the second algorithm. Projection methods introduce fewer variables but have larger degrees. We propose several adaptations to this idea to attempt to reduce the degree and number of required images. These include: sparse interpolation, evaluation point optimization and evaluating at primitive roots of unity.

We give an in depth comparison of these algorithms, both empirically and theoretically.

Acknowledgments

Completing my degree would not have been possible without the financial aid provided by NSERC, Stephen Watt and the UWO Departments of Graduate Studies and Computer Science. I would also like to thank my parents and family in Ontario for their hospitality and support. Finally, I would again like to thank my advisor Stephen Watt for his continued brilliance, patience and kindness these past years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Computer Algebra vs. Symbolic Computation

The two terms "computer algebra" and "symbolic computation" are often used interchangeably by applied mathematicians, however, it is important to note the difference between them. Symbolic computation involves the manipulation of expressions, often containing indeterminates which represent mathematical objects. Typical calculations involve equivalence, expansion of products, changing to a canonical form and simplification. Computer algebra involves computation using values from defined mathematical sets, such as polynomial rings, quotients, matrices. Typical calculations involve factorization, integrals and solving systems of equations [Wat06].

For instance, computer algebra regards the expressions $(x-1)(x+1)$ and x^2-1 as identical. Symbolic computation views the expanded and factorized polynomials as distinct. Collecting like terms is symbolic computation; a GCD computation is computer algebra [Wat07b].

Although the initial interest of researchers in this field was symbolic computation (the first features of elementary computer algebra systems was term manipulation),

the primary focus is now on efficient implementations of algorithms on specific algebraic domains. The gap between symbolic computation and computer algebra in modern computer algebra systems is increasing, and this can hinder their usability. Computer algebra often has difficulties dealing with objects of unknown size, like a system of equations of unknown size, or a polynomial of unknown variables and degree with coefficients from an unknown ring. These generalizations come naturally to us working by hand. We need a way to bridge the gap between the two concepts in mathematical computer systems. It is possible to make computer algebra more symbolic by the use of algorithms with more varied input and output domains. Conversely, symbolic computation can become more algebraic by restricting classes of allowed expressions, for example by using typed terms [Wat07b].

One such example of where computer algebraic systems are not "symbolic enough" is polynomials with unknown degree. For example, $x^2 - y^{n^2-n}$ can be factorized by hand using difference of squares, as $n^2 - n$ is always even for all integer values of n. Mathematical objects such as this, known as *symbolic polynomials*, are not supported in computer algebra systems. The purpose of this thesis is to compare several methods to find the most suitable means of providing computer algebra algorithms to symbolic polynomials.

1.2 Motivation and Related Work

Polynomials with parametric exponents arise, for example, in the study of algorithmic complexity. For instance, the number of operations for the classical traveling salesmen problem is $O(n^2 2^n)$ [DFJ54]. Moreover, there exists a type of approximation algorithm for optimizations known as *polynomial-time approximation schemes* (PTAS). All PTAS, much like quasilinear complexities, can be written as symbolic

polynomials. We see a nested symbolic polynomial in Chen's solution to the general multiprocessor job scheduling problem: $O(n^{(3mm!)^{\frac{m}{\epsilon}+1}})$ [CM99].

Yokoyama and Pan have shown how to solve systems of algebraic equations and compute Gröbner bases for polynomials with parametric exponents [Yok04, PW06]. There have also been investigations into the properties of exponential polynomials [HRS89]. Watt [Wat06, Wat07a] shows that when restricting the exponent polynomials to be integer-valued, symbolic polynomials form a UFD. He proposes two different families of algorithms for calculating factorizations, GCDs, etc. It is these two varieties of algorithms, namely change of basis and evaluation/interpolation, that we will study, extend and compare.

1.3 Outline

In this chapter, through the example of symbolic polynomials, we have shown the necessity for computer algebra systems to be "more symbolic". Symbolic polynomials arise often in practice. Modern software should be able to accommodate manipulations and computations of this type.

This rich mathematical structure and the proposed algorithms make use of a broad array of mathematical jargon and techniques. All of the relevant background information required to grasp the fundamental algorithms and analyses of this topic is found in Chapter 2. We present an introduction to algebraic domains along with many classical computer algebra strategies: interpolation, algebraic independence, gradient descent and roots of unity.

Chapter 3 provides a formal introduction to symbolic polynomials. Their ring structure is defined and is shown to form a unique factorization domain for proper coefficient rings. This is achieved by changing the exponent polynomials to the bino-

mial basis. Through algebraic independence, this is equivalent to Laurent polynomials, which are known to form a UFD.

Chapter 4 presents a means of dealing with symbolic polynomials by making use of the extension described in the previous chapter. Procedures and examples for GCDs and factorization are provided and examined. A discussion of the strengths and weaknesses of this method, in particular its inability to efficiently handle sparse input, is offered as reason for alternative methods.

Through the evaluation of exponent variables, projected images are mapped to polynomials that can be manipulated as needed and then interpolated back to the original form. Chapter 5 describes such algorithms for both dense and sparse interpolation. Projection schemes introduce a number of complications, which are described and solved. Once again, examples are provided for both GCD and factorization. Preliminary analysis details the sizable degree that may be attained when evaluated at randomly selected values, and notes the necessity for "smarter" evaluation points.

Chapter 6 illustrates two strategies to reduce the degree of evaluated images. The first performs an optimization to find the set of values that minimize the evaluations of certain polynomials. The second idea is to evaluate the exponent variables at primitive roots of unity and use the extension method to calculate each image.

Chapter 7 features empirical results for timing, degree, number of evaluations and number of variables, given for examples of the change of basis, sparse interpolation, point selection optimization and roots of unity projection algorithms.

Chapter 8 gives asymptotic complexity estimates for the algorithms in the previous chapter.

Chapter 9 concludes the thesis with a review of significant findings and possible future research prospects in this area.

Chapter 2

Preliminaries

We begin by introducing terminology needed to understand the polynomial algorithms presented in later chapters.

2.1 Interesting Domains

We say an *integral domain* is a nontrivial commutative ring R for which there are no zero divisors. In this sense, nontrivial means the multiplicative and additive identities (1 and 0 respectively) are distinct. A *zero divisor* is an element $a \in R$ such that for some other nonzero element $b \in R$, $ab = 0$. The integers form an integral domain, as does $\mathbb{Z}/p\mathbb{Z}$, where p is a prime. The polynomial ring $R[X]$ is also an integral domain if and only if the coefficient ring R is an integral domain.

An integral domain R for which there exists a *Euclidean evaluation function* $d : R \to \mathbb{N} \cup \{-\infty\}$ such that for any $a, b \in R$ if $b \neq 0$ then $\exists\, q, r \in R$ with $a = qb + r$ and $d(r) < d(b)$ is called a *Euclidean domain*. Euclidean domains allow for greatest common divisor computations, and applications of the Chinese remainder theorem [vzGG03].

Let R be a ring and $a \in R$. We define the *principal (left or right) ideal* of a as the smallest (left or right) ideal of R that has element a in it; it is denoted (a). A *principal ideal domain (PID)* is an integral domain in which all ideals are principal ideals. Some important PIDs include the rings of integers and univariate polynomials with coefficients in a field $\mathbb{F}$. Every Euclidean domain is a PID, but not conversely [MB79].

In a commutative ring R, a *unit* $u \in R$ is an element with a multiplicative inverse. There are no nonzero, nontrivial units in $\mathbb{Z}$; in $\mathbb{Z}/p\mathbb{Z}$, every nonzero element has an inverse. We say any nonzero, nonunit $p \in R$ is *reducible* if there exists elements $a, b \in R$ such that $p = ab$. If p is not reducible, then it is said to be *irreducible*. A *Unique Factorization Domain (UFD)* is an integral domain R in which every nonzero, nonunit element in R can be factored into a finite product of irreducibles that is unique (up to the reordering of terms, and multiplication of units). This notion of unique factorization is borrowed from the Fundamental Theorem of Arithmetic and generalized further. Gauss proved that if R is a UFD, then $R[X]$ is as well. Every PID, and thus every Euclidean domain, is a UFD, but not conversely [vzGG03, MB79].

Let R be a ring with $a, b, c \in R$. The element c is the *greatest common divisor* (GCD) of a and b if c divides both a and b, and for all $d \in R$ if d divides a and b then d also divides c.

A *Bézout domain* is defined as an integral domain R that has a GCD for every pair, $a, b \in R$, say $gcd(a, b)$. Moreover, $\exists\, r, s \in R$ such that

$$gcd(a, b) = ra + sb.$$

This equation is known as the Bézout identity, and the two elements r and s are the classic Bézout coefficients, found by the extended Euclidean algorithm. Every PID is

a Bézout domain, but not conversely.

An integral domain D, in which every nonzero pair $a, b \in D$ has a GCD, is known as a *GCD domain*. Clearly, both UFDs and Bézout domains are GCD domains. [MB79]

Figure 2.1 illustrates the relationship between said domains.

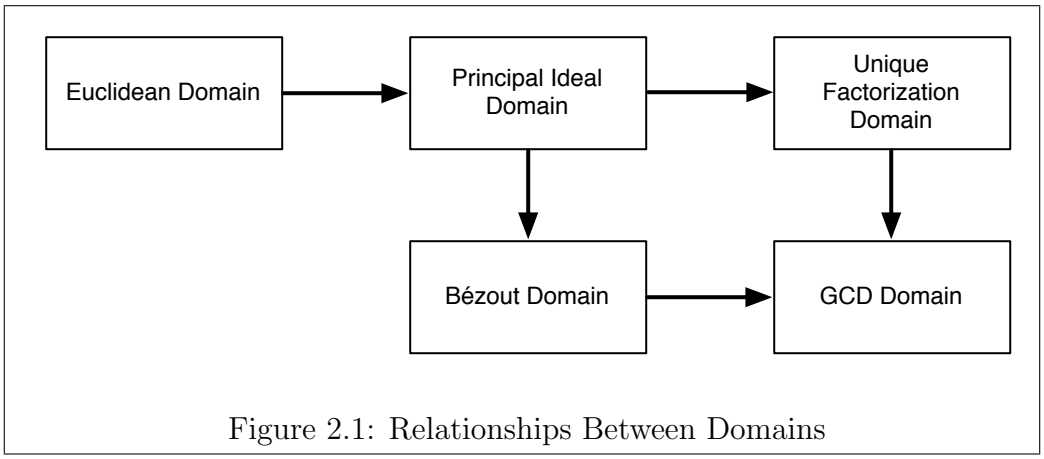


Figure 2.1: Relationships Between Domains

2.2 Polynomial Terminology

The *content* of a polynomial $f \in R[X]$, $cont(f)$, provided R is a GCD domain, is defined as the GCD of the set of coefficients, f_i; that is, $cont(f) = gcd(f_0, f_1, \ldots, f_n) \in R$. The *primitive part* of f, or $pp(f)$, is the remaining factor if $cont(f)$ is factored out. So, $f = cont(f) \cdot pp(f)$. Since R forms a UFD, the GCD computation for the content is unique up to units [vzGG03].

Example 2.2.1 Consider the following polynomial $f \in \mathbb{Z}[x]$.

$$\begin{aligned} f &= 24x^3 - 28x^2 - 12x + 4 \\ cont(f) &= gcd(24, -28, -12, 4) = 4 \\ pp(f) &= \frac{1}{4}f = 6x^3 - 7x^2 - 3x + 1 \end{aligned}$$

Notice that the coefficients of the primitive part are relatively prime and that $pp(f) \cdot cont(f) = f$.

For $f \in \mathbb{Z}[X]$, we define the *fixed divisor* $d(f)$ as the maximal positive integer that divides $f(X)$ for all possible integer values of X [Tur86].

2.3 Integer-Valued Polynomials

Let D be an integral domain, with quotient field K and $E \subseteq D$. We define

$$\mathrm{Int}(E, D)[X] = \{f(X) \in K[X] \mid f(a) \in D, \forall a \in E\} \tag{2.1}$$

as the ring of *integer-valued polynomials*. For notation simplification, we say $\mathrm{Int}(D, D)[X] = \mathrm{Int}(D)[X]$. The first investigations into $\mathrm{Int}(D)$, with D being the ring of integers in a number field, was by Polya and Ostrowski in the early twentieth century [Wat06]. In the past twenty years there has been much more interest in $\mathrm{Int}(E, D)$. These recent works have focused on Dedekind domains, Krull Rings and Prüfer domains [GHLS90, Fri96, Cha93, CCS98].

For our purposes we are only interested in $\mathrm{Int}(\mathbb{Z})[X]$. This is the ring of polynomials whose evaluation must give an integer at all integers. For instance, every polynomial with integer coefficients, $\mathbb{Z}[X]$, is also in $\mathrm{Int}(\mathbb{Z})[X]$.

Example 2.3.1 Consider $p(n) = \frac{1}{2}n^2 + \frac{1}{2}n$. As $n^2 + n$ is always even, $p(n)$ must always evaluate to an integer and is thus in the ring of integer-valued polynomials.

It is well known that the *binomial polynomials*, $\binom{x}{n} = \frac{x(x-1)\ldots(x-n+1)}{n!}$ form a basis for $\mathrm{Int}(\mathbb{Z})$ [GHLS90]. Here are the first five binomial polynomials and their monomial representation:

$$\binom{x}{0} = 1 \tag{2.2}$$

$$\binom{x}{1} = x \tag{2.3}$$

$$\binom{x}{2} = \frac{1}{2}x^2 - \frac{1}{2}x \tag{2.4}$$

$$\binom{x}{3} = \frac{1}{6}x^3 - \frac{1}{2}x^2 + \frac{1}{3}x \tag{2.5}$$

$$\binom{x}{4} = \frac{1}{24}x^4 - \frac{1}{4}x^3 + \frac{11}{24}x^2 - \frac{1}{4}x \tag{2.6}$$

$$\binom{x}{5} = \frac{1}{120}x^5 - \frac{1}{12}x^4 + \frac{7}{24}x^3 - \frac{5}{12}x^2 + \frac{1}{5}x \tag{2.7}$$

A very useful property of using the binomial basis to represent an integer-valued polynomial is that it is easy to calculate its fixed divisor. The fixed divisor is actually the GCD of the basis coefficients. We give a simple example to illustrate this notion.

Example 2.3.2 Consider $f(x) = x^2 + x$, with $f \in \mathrm{Int}(\mathbb{Z})$. Rewritting f in the binomial basis we have $f_b(x) = 2\binom{x}{2} + 2\binom{x}{1}$. The coefficients of both terms of the binomial are 2, thus $d(f) = 2$. This is the expected answer, as f always evaluates to an even number.

2.4 Interpolation

Given a set of data points, we can use *interpolation* to find a polynomial that passes through all those points. It is a very useful tool in both numerical analysis and computer algebra, but the way it is used is quite different in the two fields. Many computer algebra algorithms make use of evaluation and interpolation to work with a more computationally efficient structure (moving from polynomials to the integers or

rationals for instance). The desired result can be reconstructed using interpolation.

Suppose we wish to interpolate to reconstruct a univariate polynomial, $f = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ of degree n. To do this, we need values for the $n+1$ coefficients, $a_0, \ldots, a_n$. We require $n+1$ points, $u_0, u_1, \ldots, u_n$ and evaluations $f(u_i) = y_i$ for i from 0 to n. From these points we can create what is referred to as a *Vandermonde Matrix* [vzGG03]

$$VDM(u_0, u_1, \ldots, u_n) = \begin{pmatrix} 1 & u_0 & u_0^2 & \ldots & u_0^{n+1} \\ 1 & u_1 & u_1^2 & \ldots & u_1^{n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u_n & u_n^2 & \ldots & u_n^n \end{pmatrix} \tag{2.8}$$

Note that there is one column for every possible term of the interpolated polynomial. It is possible that the points used for evaluation can create linear dependencies, however, when nonsingular, we can use the Vandermonde matrix to set up a system of linear equations, which can be solved to give the coefficients of the goal polynomial,

$$VDM(u_0, u_1, \ldots, u_n) \cdot \begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}. \tag{2.9}$$

Example 2.4.1 Given the following points, interpolate for a polynomial $f(x)$ of degree 2.

x	$f(x)$
1	7
2	12
3	19

Table 2.1: Evaluation Points for Example 2.4.1

We can construct the Vandermonde system of linear equations.

$$\begin{pmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 7 \\ 12 \\ 19 \end{pmatrix} \tag{2.10}$$

This gives us $a_2 = 1, a_1 = 2, a_0 = 4$, and the polynomial $f(x) = x^2 + 2x + 4$.

Using Gaussian elimination, this requires $O(n^3)$ operations. There are also other classical interpolation methods, such as Newton and Lagrange which can reduce this to $O(n^2)$. Fast arithmetic via the Fast Fourier Transform (FFT) can compute interpolations in $O(n \log^2(n) \log \log n)$[vzGG03].

The Vandermonde system can be used in the generalization to the multivariate case. In doing so, there must be entries in the matrix for every possible term in the resulting polynomial.

Example 2.4.2 Say we are looking for a bivariate polynomial, with each separate variable having a degree of no more than 2. This implies that the resulting polynomial is of the form

$$f(x, y) = a_8x^2y^2 + a_7x^2y + a_6x^2 + a_5xy^2 + a_4xy + a_3x + a_2y^2 + a_1y + a_0.$$

We need 9 independent evaluations to solve the system. After selecting the following evaluations, we can set up the extended Vandermonde system.

x	y	$f(x,y)$
-9	1	244
-9	-6	11319
10	10	39776
-10	-2	1556
10	-10	39376
4	10	6014
-10	5	9711
-4	6	2054
4	-1	85

Table 2.2: Evaluation Points for Example 2.4.2.

$$\begin{pmatrix} 81 & 81 & 81 & -9 & -9 & -9 & 1 & 1 & 1 \\ 2916 & -486 & 81 & -324 & 54 & -9 & 36 & -6 & 1 \\ 10000 & 1000 & 100 & 1000 & 100 & 10 & 100 & 10 & 1 \\ 400 & -200 & 100 & -40 & 20 & -10 & 4 & -2 & 1 \\ 10000 & -1000 & 100 & 1000 & -100 & 10 & 100 & -10 & 1 \\ 1600 & 160 & 16 & 400 & 40 & 4 & 100 & 10 & 1 \\ 2500 & 500 & 100 & -250 & -50 & -10 & -25 & 5 & 1 \\ 576 & 96 & 16 & -144 & -24 & -4 & 36 & 6 & 1 \\ 16 & -16 & 16 & 4 & -4 & 4 & 1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_8 \\ a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 244 \\ 11319 \\ 39776 \\ 1556 \\ 39376 \\ 6014 \\ 9711 \\ 2054 \\ 85 \end{pmatrix} \tag{2.11}$$

This gives us

$$\begin{pmatrix} a_8 \\ a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 0 \\ 0 \\ 2 \\ 7 \\ -5 \\ 0 \\ 6 \end{pmatrix} \tag{2.12}$$

and the polynomial $f(x, y) = 4x^2y^2 + 2xy + 7x - 5y^2 + 6$.

This simple extension for multivariate polynomials comes at a cost. For a polynomial of v variables and degree d, this requires $O((d+1)^v)$ evaluation points. This exponential factor does not scale well for polynomials of many variables and larger degrees. It is particularly wasteful when dealing with sparse polynomials. The interpolation of $x^{99}y^{99} + 1$ would require on the order of 10000 evaluations, even though the polynomial only has a pair of terms! In practice, the polynomials that we are interested in are sparse. We will exploit this by using a more intelligent sparse interpolation method.

In 1979, Zippel proposed a probabilistic sparse black-box interpolation as part of his PhD thesis. For a *t-sparse* polynomial (has t terms and $t \ll (d+1)^v$), this can reduce the number of evaluations to $O(vdt)$. Our goal polynomial is $P(x_1, x_2, \ldots, x_v)$, with an initial evaluation at $(e_{1,1}, e_{2,1}, \ldots, e_{v,1})$. The algorithm proceeds by interpo-

lating for one variable at each step. This produces the following polynomials

$$P_1 = P(x_1, e_{2,1}, \ldots, e_{v,1}) \tag{2.13}$$

$$P_2 = P(x_1, x_2, e_{3,1}, \ldots, e_{v,1}) \tag{2.14}$$

$$\vdots \qquad \vdots \tag{2.15}$$

$$P_v = P(x_1, x_2, \ldots, x_v) \tag{2.16}$$

At the end of the first step, P_1 is a univariate polynomial with the indeterminate x_1. Every coefficient of P_1 is a polynomial with the following indeterminates: $x_2, \ldots, x_v$. We denote the k^{th} coefficient by $f_k(x_2, \ldots, x_v)$. The probabilistic assumption, and key to the algorithm, is that if a coefficient interpolates to zero, it is zero everywhere. There are two possible reasons why a term, say the k^{th} term, may not be present in P_1. Either f_k is identically zero or $f_k(e_{2,1}, \ldots, e_{v,1}) = 0$. We can control the values $e_{i,j}$ – if they come from a large random set of integers the probability that $f_k(e_{2,1}, \ldots, e_{v,1})$ is zero, for $f_k \neq 0$, is very small. By implication, there is a very high probability that f_k must be zero, and that the k^{th} term does not appear in our final goal polynomial, P [Zip79].

Once it is discovered that certain terms are zero, those columns can be removed from the linear system, allowing it to be solved with fewer evaluation points. The algorithm is presented in Figure 2.2.

As input, this algorithm accepts:

- p, a v-variable black-box function.
- d, the maximum degree of $x_1, x_2, \ldots, x_v$.

The algorithm begins by performing an initial $d + 1$ evaluations. Only the value of the first variable is changed in these calculations. We can then interpolate for

zippel_interp:
Input: $p : x_1, x_2, \ldots, x_v \mapsto R, d \in \mathbb{Z}$
Output: $r \in R[x_1, \ldots, x_v]$ such that r is an interpolating polynomial over evaluations of p

1. Let $e_{i,j}$ be the i^{th} evaluation for x_j
2. $r_1 \leftarrow p(e_{1,1}, \ldots, e_{1,v})$
3. $t \leftarrow 1$
4. For i from 1 to v do
   (a) For j from 2 to $d+1$ do
      i. For k from 1 to t do
         A. $s_k \leftarrow e_{1,k}, \ldots, e_{i-1,k}, e_{i,j}, e_{i+1,1}, \ldots, e_{v,1}$
         B. $p'_k \leftarrow p(s_k)$
      ii. Using p'_k and s_k, let r_j be the polynomial with the form of r_1 whose coefficients are the solution of the system of t linear equations
   (b) Let $r_1 \in R[x_1, \ldots, x_i]$ be the interpolating polynomial over the points $(e_{i,1}, r_1), (e_{i,2}, r_2), \ldots, (e_{i,d+1}, r_{d+1})$
   (c) Let t be the number of terms in r_1
5. Return r_1

Figure 2.2: Zippel Sparse Interpolation Algorithm

that variable. From this interpolated polynomial, P_1, we must create a skeleton, or anchor. This is done by replacing the coefficients with indeterminates. For instance, assuming integer coefficients, if $P_1 = 4X_1^5 - X_1^2 + 7$, then we can be most positive that the goal polynomial is of the form $AX_1^5 + BX_1^2 + C$, where $A, B, C \in \mathbb{Z}[X_2, \ldots, X_v]$. We need another d polynomials of the same form as this anchor to interpolate for the next variable. Fortunately, now we only need t points to solve for these polynomials, rather than $d+1$. That is what is being done in the inner-most loop of the algorithm. The algorithm is complete when this is repeated v times.

Example 2.4.3 Perform Zippel interpolation to find the bivariate polynomial with individual variable degree no higher than 5. We will use the evaluations (but not necessarily all of them) found in Table 2.3.

x	y	$P(x,y)$	x	y	$P(x,y)$
1	1	-6	4	1	6147
1	2	-24	4	2	49095
1	3	-170	4	3	165571
1	4	-648	4	4	392199
1	5	-1758	4	5	765507
1	6	-3896	4	6	1321927
2	1	183	5	1	18762
2	2	1467	5	2	149952
2	3	4855	5	3	505942
2	4	11259	5	4	1198992
2	5	21495	5	5	2341266
2	6	36283	5	6	4044832
3	1	1454	6	1	46679
3	2	11600	6	2	373211
3	3	39042	6	3	1259415
3	4	92288	6	4	2984987
3	5	179750	6	5	5829527
3	6	309744	6	6	10072539

Table 2.3: Evaluation Points for Example 2.4.3.

The first step is to do an initial evaluation/interpolation for one variable to find its structure. We will interpolate for x by keeping the y value constant at $y = 1$. This gives us the following $(x, P(x, 1))$ pairs to interpolate: (1,-6), (2,183), (3, 1454), (4,6147), (5,18762), (6,46679). In doing so, we obtain the polynomial

$$(y = 1) \quad P_1 = 6x^5 + x^2 - 13. \tag{2.17}$$

Thus, our anchor is $Ax^5 + Bx^2 + C$, with only three terms. To interpolate for y, we will need five more polynomials like P_1. Due to the (in this case) sound assumption of reappearing zero coefficients, we only need three evaluations to produce such

polynomials.

Setting $y = 2$, we have the following $(x, P(x, 2))$ pairs: $(1, -24), (2, 1467)$, $(3, 11600)$. Any three random x values would suffice, the first three were chosen for consistency. We can substitute these x values into the anchor to obtain a system of linear equations

$$A + B + C = -24 \tag{2.18}$$

$$32A + 4B + C = 1467 \tag{2.19}$$

$$243A + 9B + C = 11600 \tag{2.20}$$

which can be easily solved to return $\{A = 48, B = 1, C = -73\}$ and substituted back into the anchor to get

$$(y = 2) \qquad 48x^5 + x^2 - 73 \tag{2.21}$$

as our second polynomial needed to interpolate for y. This same process of substituting three random x values and solving the system of equations can be done for $y = 3, 4, 5, 6$ to obtain

$$(y = 3) \qquad 162x^5 + x^2 - 333 \tag{2.22}$$

$$(y = 4) \qquad 384x^5 + x^2 - 1033 \tag{2.23}$$

$$(y = 5) \qquad 750x^5 + x^2 - 2509 \tag{2.24}$$

$$(y = 6) \qquad 1296x^5 + x^2 - 5193 \tag{2.25}$$

Along with equations 2.17 and 2.21 we can now interpolate for y. Interpolation of each coefficient is independent of the others. To interpolate the x^5 coefficient we

have the following pairs of points:

$$(1,6),(2,48),(3,162),(4,384),(5,750),(6,1296),$$

which the polynomial $6y^3$ passes through. The coefficient of the x^2 term is trivially 1. For the constant term, the following pairs of points

$$(1,-13),(2,-13),(3,-333),(4,-1033),(5,-2509),(6,-5193)$$

gives us the polynomial $-4y^4 - 9$. Setting these values to A, B, C of the anchor respectively produces our final answer

$$6x^5y^3 + x^2 - 4y^4 - 9 \tag{2.26}$$

Zippel's sparse interpolation required only 21 evaluations, while the naive multivariate Vandermonde method would require 36.

2.5 Algebraic Independence

Let L be a field and K a field extension of L. A sequence of distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_n \in L$ are said to be *algebraically independent* over K if every non-trivial polynomial $P(x_1, x_2, \ldots, x_n)$ in $K[x_1, x_2, \ldots, x_n]$ satisfies

$$P(\alpha_1, \alpha_2, \ldots, \alpha_n) \neq 0. \qquad \text{[MB79]}$$

This definition extends from fields to where K is a ring. A key notion of our mathematical description of symbolic polynomials requires the algebraic independence of

$x^n, x^{n^2}, x^{n^3}, \ldots$, which we will now prove.

Theorem 2.5.1 [Wat06] *The symbolic monomials $x^n, x^{n^2}, \ldots$ are algebraically independent over the base coefficient ring R.*

Proof: We will prove this by showing the algebraic independence of any two of these monomials, x^{n^a} and x^{n^b} with $a \neq b$, which then can be generalized further. For this pair to be algebraically independent there can not exist a polynomial P of any fixed degree d, with coefficients in R such that

$$P(x^{n^a}, x^{n^b}) = 0$$

Substituting the symbolic monomials into such a polynomial would produce

$$\begin{aligned} P(x^{n^a}, x^{n^b}) &= \sum_{i,j=0}^{d} C_{i,j}(x^{n^a})^i(x^{n^b})^j \\ &= \sum_{i,j=0}^{d} C_{i,j}x^{in^a}x^{jn^b} \\ &= \sum_{i,j=0}^{d} C_{i,j}x^{in^a+jn^b} \end{aligned}$$

This is a polynomial with $(d+1)^2$ possible terms. For P to evaluate to zero, either every term is identically zero, that is, each $C_{i,j} = 0$, or there is a combining of like terms and a simplification. The first case is trivial and disregarded. The second case is also not possible as there can not be any like terms to combine. $C_{i,j}x^{in^a+jn^b}$ can only combine additively with $C_{k,m}x^{kn^a+mn^b}$ if $(i,j) = (k,m)$.

2.6 Gradient Descent

Gradient Descent, or *Steepest Descent* is a classical method used in finding the local minima of nonlinear functions. The *gradient* of f, $\overrightarrow{\nabla} f$, is a vector field, whose components are the partial derivatives of f.

$$\overrightarrow{\nabla} f(x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) \tag{2.27}$$

The gradient, $\overrightarrow{\nabla} f$, is a vector in the direction of most rapid increase; therefore, $-\overrightarrow{\nabla} f$ is a vector in the direction of most rapid decrease. Figure 2.3 uses this technique to locate a local minimum of a function. It takes an initial guess, and finds the vector of most rapid decrease. A fixed step is taken in that direction. If f evaluates to a smaller value, then it is accepted as the next approximate solution. If f evaluates to a larger value, it has stepped over the local minimum, and a smaller step must be taken. The algorithm continues for a set number of iterations, or until it has converged to the minimum point. Convergence occurs when the difference between the last two neighboring approximate solutions are within a certain very small tolerance [Fau03].

2.7 Roots of Unity

An n^{th} *root of unity* is a solution of

$$\omega^n = 1 \quad \text{for } n \in \mathbb{Z}^+,\ \omega \in \mathbb{C}$$

steepest_descent:
Input: $f \in R[x_1, \ldots, x_n], p_0 \in \mathbb{Q}^n$
Output: $p \in \mathbb{Q}^n s.t. f(p)$ is a minumum

1. Let m be the maximum number of iterations
2. Let ϵ be the tolerance of covergence
3. For i from 1 to m do
   (a) $p_i \leftarrow p_{i-1}$
   (b) $dx \leftarrow -\overrightarrow{\nabla} f(p_i)$
   (c) $z_0 \leftarrow f(p_i)$
   (d) $z_1 \leftarrow f(p_i + dx)$
   (e) while $z_1 - z_0 \geq 0$ do
      i. $dx \leftarrow dx/2$
      ii. $z_1 \leftarrow f(p_1 + dx)$
   (f) if $|z_1 - z_0| < \epsilon$ then break
4. return p_i

Figure 2.3: Minimization by Steepest Descent [Fau03]

In general ω can be an element of a ring R, but for our purposes a complex ω is sufficient. An n^{th} root of unity is called *primitive* if

$$\omega^k \neq 1 \quad \text{for } k = 1, \ldots, n-1$$

One primitive n^{th} root of unity can be calculated as follows

$$\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i \sin(2\pi/n) \tag{2.28}$$

There are n n^{th} roots of unity in $\mathbb{C}$. For instance, the two square roots of unity are 1 and -1. Figure 2.4 contains the fifth roots of unity in the complex plane. Given

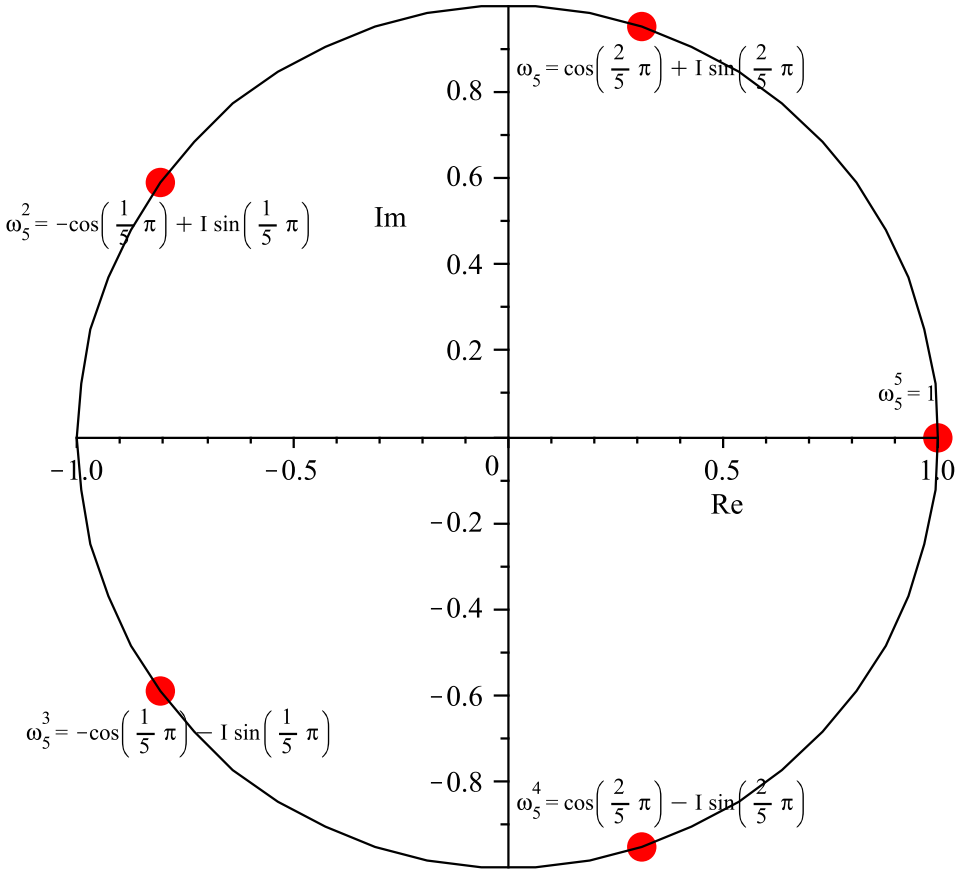


Figure 2.4: Fifth Roots of Unity in the Complex Plane

a primitive root of unity, ω_n from Equation 2.28, the remaining roots are its powers. That is to say, the n^{th} roots of unity are

$$\omega_n^k = e^{2k\pi i/n} = \cos(2k\pi/n) + i\sin(2k\pi/n) \quad \text{for } k = 0, \ldots, n-1$$

The sequence of powers ω_n^k is periodic, repeating every n elements. The cyclical nature of roots of unity is best illustrated in Figure 2.4. The next power of ω_5 is obtained by following the unit circle counter clockwise until it meets the next root.

In other words,

$$\omega_n^i \quad = \quad \omega_n^{i+kn} \quad \text{for } i < n,\ k \in \mathbb{Z}$$

When n is a prime number, the n^{th} roots of unity have some interesting properties. Besides the trivial root ω_n^0, every root is a primitive root of unity. Table 2.4 contains the powers of the fifth roots of unity. The table is symmetric, and each primitive root generates the values in a unique order.

ω_5^1	$(\omega_5^1)^1 = \omega_5^1$	$(\omega_5^1)^2 = \omega_5^2$	$(\omega_5^1)^3 = \omega_5^3$	$(\omega_5^1)^4 = \omega_5^4$
ω_5^2	$(\omega_5^2)^1 = \omega_5^2$	$(\omega_5^2)^2 = \omega_5^4$	$(\omega_5^2)^3 = \omega_5^1$	$(\omega_5^2)^4 = \omega_5^3$
ω_5^3	$(\omega_5^3)^1 = \omega_5^3$	$(\omega_5^3)^2 = \omega_5^1$	$(\omega_5^3)^3 = \omega_5^4$	$(\omega_5^3)^4 = \omega_5^2$
ω_5^4	$(\omega_5^4)^1 = \omega_5^4$	$(\omega_5^4)^2 = \omega_5^3$	$(\omega_5^4)^3 = \omega_5^2$	$(\omega_5^4)^4 = \omega_5^1$

Table 2.4: Powers of the Fifth Roots of Unity

Chapter 3

Symbolic Polynomials

This chapter will introduce the reader to symbolic polynomials, provide a ring structure and prove that this ring is a UFD.

3.1 Introduction and Problem Statement

Symbolic polynomials, whose exponents are not known in advance, are a rich and interesting mathematical structure. For our purposes, these unknown powers are represented as polynomials themselves [Wat06]. Every exponent, rather than being a nonnegative integer, is a polynomial. The exponent polynomials may or may not be independent of the base. The following are some examples of symbolic polynomials.

$$5x^{2n^2+6mn-7n+1} - 8x^{5n^2}$$

$$x^{7n-1}y^{n^2-9} + 16x^{n^2}y^{8n+4}$$

$$12nx^{7n-1}y^{n^2-9} + 16x^{n^2}y^{8n+4}$$

The first two polynomials have base variables x, y and exponential variables m, n.

The final example has n in both the base and exponent. This is still a valid symbolic polynomial; however, this sharing of variables is a more complicated notion and causes problems with one of our algorithms.

Naturally, we are interested in performing mathematical operations with these objects. Some elementary operations are trivial. Addition, multiplication and derivatives are performed as one would expect.

Example 3.1.1 Simplify the following symbolic polynomial through addition by combining like terms.

$$\begin{aligned} P &= 2x^{n^3} + 4x^{n^2} - 2x^{2n^2} + x^{n^3} \\ &= 3x^{n^3} + 4x^{n^2} - 2x^{2n^2} \end{aligned}$$

Terms only combine additively if the exponent polynomials are exactly the same.

Example 3.1.2 Simplify the following symbolic polynomial through multiplication.

$$\begin{aligned} P &= (2x^{n^3} + 4x^{n^2}) \cdot (2x^{2n^2} - x^{n^3}) \\ &= 2x^{n^3}2x^{2n^2} - 2x^{n^3}x^{n^3} + 4x^{n^2}2x^{2n^2} - 4x^{n^2}x^{n^3} \\ &= 4x^{n^3+2n^2} - 2x^{2n^3} + 8x^{3n^2} - 4x^{n^3+n^2} \end{aligned}$$

After expanding the polynomial, the powers of the same base for every term are added together.

There are other crucial operations that are not so easily executed. The next two chapters give a pair of algorithms which can be used for these more computationally difficult operations, such as GCD and factorization.

3.2 A Ring Structure for Symbolic Polynomials

In searching for a meaningful representation of this structure, several notions come to mind. If the polynomial P has only a single exponent variable, n, then we could say P is the set of polynomials for all values of n. However, this is not very intuitive. Another idea is to say that a set S under a mapping, ϕ, that produces a polynomial $R[x_1, x_2, \ldots, x_n]$ completely describes symbolic polynomials. This definition is far too general. A ring structure is needed.

Watt [Wat06] defines *the ring of symbolic polynomials* with base variables $x_1, x_2, \ldots, x_v$ over the coefficient ring R and symbolic exponents in $n_1, n_2, \ldots, n_p$ as the ring of finite sums of the following form

$$\sum_i c_i x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_v^{e_{iv}}$$

where $c_i \in R$ and $e_{ij} \in \mathrm{Int}(\mathbb{Z})[n_1, n_2, \ldots, n_p]$, an integer-valued polynomial. Multiplication is defined, as expected and demonstrated from Example 3.1.2,

$$c_1 x_1^{e_{11}} x_2^{e_{12}} \cdots x_n^{e_{1n}} \times c_2 x_1^{e_{21}} x_2^{e_{22}} \cdots x_n^{e_{2n}} = c_1 c_2 x_1^{e_{11}+e_{21}} x_2^{e_{12}+e_{22}} \cdots x_n^{e_{1n}+e_{2n}}$$

The ring of symbolic polynomials is denoted $R[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v]$.

If we evaluate the symbolic exponent variables at integers, we have

$$\phi : R[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v] \to R[x_1, x_2, \ldots, x_v, x_1^{-1}, x_2^{-1}, \ldots, x_v^{-1}].$$

That is, evaluating the exponents at integers gives us *Laurent polynomials*. A Laurent polynomial allows for negative exponents. This mapping is valid due to our restriction of the symbolic exponents to integer-valued polynomials [Wat06, Wat07a].

Example 3.2.1 Consider the polynomial $p(n;x) = 2x^n + 3x^{n^2}$. Let us evaluate n at two points.

$$\begin{aligned} p(2;x) &= 2x^2 + 3x^4 \\ p(-2;x) &= 2x^{-2} + 3x^4 \end{aligned}$$

Evaluating the symbolic polynomial at $n = -2$ gives a negative exponent. We could put a restriction on the symbolic exponents to only evaluate to positive integers, but we find this to be more unwieldy than working with Laurent polynomials.

3.3 Symbolic Polynomial Rings are UFDs

Two of the most important operations we wish to perform on symbolic polynomials are GCDs and factorizations. We must show that $R[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v]$ is a UFD, and thus a GCD domain. This will be done by proving that they can be transformed to Laurent polynomials, which form a UFD. For simplicity, we take $R = \mathbb{Q}$.

Theorem 3.3.1 [Wat06] $\mathbb{Q}[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v]$ *is a UFD, and monomials are units.*

Proof: We begin by restricting our exponents to come from $\mathbb{Z}[n_1, \ldots, n_p]$. We will show that our symbolic polynomials are isomorphic to a form that we are familiar with and that has a UFD structure. Consider the following manipulation on the

monomial $x_k^{e_{ij}}$.

$$\begin{aligned} x_k^{e_{ij}} &= x_k^{\sum_j h_{ij} n_1^j} && \text{for } h_{ij} \in \mathbb{Z}[n_2, \ldots, n_p] \\ &= \prod_j (x_k^{n_1^j})^{h_{ij}} \end{aligned}$$

By changing the monomial to a product, we have isolated n_1 from the remaining exponent variables. As $x_k^{n_1}, x_k^{n_1^2}, \ldots$ are algebraically independent, we may replace each $x_k^{n_1^j}$ with an independent variable. Using the change of variables, $x_k^{n_1^j} \to x_{k1j}$, we produce the following isomorphism

$$\mathbb{Q}[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v] \cong$$
$$\mathbb{Q}[n_2, n_3, \ldots, n_p; x_{110}^{\pm 1}, x_{111}^{\pm 1}, \ldots, x_{210}^{\pm 1}, x_{211}^{\pm 1}, \ldots, x_{v10}^{\pm 1}, x_{v11}^{\pm 1}, \ldots]$$

The variable n_1 is no longer an exponent variable, and in its place are additional base variables. We can inductively follow this procedure an additional $p-1$ times to eliminate every exponent indeterminate, thus producing

$$\mathbb{Q}[n_1, n_2, \ldots, n_p; x_1, x_2, \ldots, x_v] \cong$$
$$\mathbb{Q}[; x_{110}^{\pm 1}, \ldots, x_{v1\ldots}^{\pm 1}, x_{120}^{\pm 1}, \ldots, x_{v2\ldots}^{\pm 1}, \ldots, x_{1p0}^{\pm 1}, \ldots, x_{vp\ldots}^{\pm 1}]$$

This is a Laurent polynomial ring, which is a UFD and has monomials as units. If we were to remove our above assumption and allow for integer-valued polynomials, we would have a similar construction; however, fixed divisors must be dealt with. As we have shown in the previous chapter, the fixed divisor is the GCD of the set of coefficients when the polynomial is rewritten in the binomial basis. As such, the change of variables will become $x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \to x_{k,i_1\ldots i_p}$.

The following chapter on the change of basis extension algorithm will give some examples that illustrate the necessity for the change to the binomial basis.

Chapter 4

Change of Basis Algorithms

This chapter introduces the first algorithm allowing UFD operations on symbolic polynomials.

4.1 Motivation

Watt's change of basis extension algorithm uses the ideas from Theorem 3.3.1 to change the symbolic exponents to the binomial basis. After a change of variables, the resulting Laurent polynomial can be manipulated as one sees fit, and then mapped back to its original form [Wat06]. The following two examples will help illustrate the binomial mapping $\gamma : x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \mapsto x_{k,i_1\ldots,i_p}$.

Example 4.1.1 Change to the binomial basis, and apply the above mapping to $f(n;x) = 4x^{n^2+n} - x^{n^2+3}$. Using Equations 2.2 – 2.4 we can solve for x^2 (1 and x are

trivial). We list the first three powers of x in binomial basis.

$$1 = \binom{n}{0} \tag{4.1}$$

$$n = \binom{n}{1} \tag{4.2}$$

$$n^2 = 2\binom{n}{2} + \binom{n}{1} \tag{4.3}$$

After performing the above substitution to f,

$$\begin{aligned} f(n;x) &= 4x^{2\binom{n}{2}+2\binom{n}{1}} - x^{2\binom{n}{2}+\binom{n}{1}+3\binom{n}{0}} \\ &= 4x^{2\binom{n}{2}}x^{2\binom{n}{1}} - x^{2\binom{n}{2}}x^{\binom{n}{1}}x^{3\binom{n}{0}} \end{aligned}$$

the new variables introduced through the mapping γ are as follows

$$\begin{aligned} A &= x^{\binom{n}{0}} \\ B &= x^{\binom{n}{1}} \\ C &= x^{\binom{n}{2}} \end{aligned}$$

After a substitution we get our goal Laurent polynomial of

$$f(A,B,C) = 4C^2B^2 - C^2BA^3$$

We now show an example of a multivariate mapping.

Example 4.1.2 Change to the binomial basis, and apply the above mapping to $f(n, m; x) = x^{n^2m^2+2m-n}$. Using Equations 4.1 – 4.3, f becomes

$$\begin{aligned} f(n,m;x) &= x^{(2\binom{n}{2}+\binom{n}{1})(2\binom{m}{2}+\binom{m}{1})+2\binom{n}{0}\binom{m}{1}-\binom{n}{1}\binom{m}{0}} \\ &= x^{4\binom{n}{2}\binom{m}{2}+2\binom{n}{2}\binom{m}{1}+2\binom{n}{1}\binom{m}{2}+\binom{n}{1}\binom{m}{1}+2\binom{n}{0}\binom{m}{1}-\binom{n}{1}\binom{m}{0}} \\ &= x^{4\binom{n}{2}\binom{m}{2}}x^{2\binom{n}{2}\binom{m}{1}}x^{2\binom{n}{1}\binom{m}{2}}x^{\binom{n}{1}\binom{m}{1}}x^{2\binom{n}{0}\binom{m}{1}}/x^{\binom{n}{1}\binom{m}{0}} \end{aligned}$$

The new variables introduced through the mapping γ are as follows:

$$\begin{aligned} x_{1,0} = A &= x^{\binom{n}{1}\binom{m}{0}} \\ x_{0,1} = B &= x^{\binom{n}{0}\binom{m}{1}} \\ x_{1,1} = C &= x^{\binom{n}{1}\binom{m}{1}} \\ x_{2,1} = D &= x^{\binom{n}{2}\binom{m}{1}} \\ x_{1,2} = E &= x^{\binom{n}{1}\binom{m}{2}} \\ x_{2,2} = F &= x^{\binom{n}{2}\binom{m}{2}} \end{aligned}$$

After a substitution we get our goal polynomial of

$$f(A, B, C, D, E, F) = F^4D^2E^2CB^2A^{-1}$$

4.2 Extension Algorithms

The extension algorithm for GCD is shown in Figure 4.1. It takes as input two symbolic polynomials p and q, outputting their GCD g [Wat06].

The notion of applying GCDs to symbolic polynomials can be difficult to perceive. For an evaluation map on the symbolic exponents $\phi : \text{Int}[n_1, \ldots, n_p](\mathbb{Z}) \to \mathbb{Z}$,

cob_gcd:
Input: $p, q \in R[n_1, \ldots, n_p; x_1, \ldots, x_v]$
Output: $g \in R[n_1, \ldots, n_p; x_1, \ldots, x_v]$ such that g is the GCD of p and q

1. Let p' be the result of changing the exponent polynomials of p to the binomial basis
2. Let q' be the result of changing the exponent polynomials of q to the binomial basis
3. $\gamma : x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \mapsto x_{k,i_1\ldots,i_p}$
4. $P \leftarrow \gamma(p')$
5. $Q \leftarrow \gamma(q')$
6. $G \leftarrow \gcd(P, Q)$
7. $g \leftarrow \gamma^{-1}(G)$
8. Return g after changing the exponent polynomials back to a power basis

Figure 4.1: Change of Basis Extension Algorithm for GCDs of Symbolic Polynomials

$\phi(g) \mid \gcd(\phi(p), \phi(q))$. For other polynomials $g' \in \mathbb{Q}[n_1, \ldots, n_p; x_1, \ldots, x_v]$ such that $\phi(g') \mid \phi(p)$ and $\phi(g') \mid \phi(q)$, then $g' \mid g$, making g the greatest common divisor. This computation is unique up to units, being monomials.

The first step of the extension algorithm is to convert the exponents of the two input polynomials to the binomial basis. This is done in the same manner as shown in Example 4.1.1 and Example 4.1.2. P and Q are then constructed using the mapping γ, with d_i being the maximum degree of n_i in p and q. These newly constructed polynomials are Laurent polynomials. Their GCD is then found and it is remapped to its symbolic binomial exponent form, and then switched back to the power basis.

The extension algorithm for factorization is shown in Figure 4.2. It takes as input symbolic polynomial p, outputting $f_1, \ldots, f_m$ such that their product is equal to p,

cob_factorize:
Input: $p \in R[n_1, \ldots, n_p; x_1, \ldots, x_v]$
Output: $f_1, \ldots, f_m \in R[n_1, \ldots, n_p; x_1, \ldots, x_v]$ such that all f_i are irreducible and their product is p

1. Let p' be the result of changing the exponent polynomials of p to the binomial basis
2. $\gamma : x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \mapsto x_{k,i_1\ldots,i_p}$
3. $P \leftarrow \gamma(p')$
4. $F_1, \ldots, F_m \leftarrow \text{factor}(P)$
5. For i from 1 to m do
   (a) $f_i \leftarrow \gamma^{-1}(F_i)$
6. Return $f_1, \ldots, f_m$ after changing the exponent polynomials back to a power basis

Figure 4.2: Change of Basis Extension Algorithm for Factorization of Symbolic Polynomials

unique to units. [Wat06]

4.3 The Necessity for Binomial Basis

We have noted that because of fixed divisors, it is necessary to change the exponents to a binomial basis. The reason behind this is not always clear to the reader. If $x^n, x^{n^2}, \ldots$ are algebraically independent, one might think that a simple substitution would suffice. However, that can eliminate the shared properties of the different terms. For instance, consider Example 4.3.1. By performing the incorrect variable substitution, we lose the property that both exponents when evaluated must always be even, and the difference of squares could not be used.

Example 4.3.1 Factorize $f(n;x,y) = x^{n^2-n} - y^2$. First, let us try to factorize without changing to the binomial basis. By using the substitution

$$\begin{aligned} A &= x^{n^2} \\ B &= x^n \\ C &= y \end{aligned}$$

we get $f(A,B,C) = AB^{-1} - C^2$, which has no smaller factors. However, if we performed our change of basis first, we would have

$$\begin{aligned} f(n;x,y) &= x^{2\binom{n}{2}} - y^{2\binom{n}{0}} \\ &= (x^{\binom{n}{2}})^2 - (y^{\binom{n}{0}})^2 \end{aligned}$$

Substituting the following

$$\begin{aligned} A &= x^{\binom{n}{2}} \\ B &= y^{\binom{n}{0}} \end{aligned}$$

we have

$$\begin{aligned} f(A,B) &= A^2 - B^2 \\ &= (A-B)(A+B) \\ f(n;x,y) &= (x^{\binom{n}{2}} - y^{\binom{n}{0}})(x^{\binom{n}{2}} + y^{\binom{n}{0}}) \\ &= (x^{(n^2-n)/2} - y)(x^{(n^2-n)/2} + y) \end{aligned}$$

4.4 GCD and Factorization Examples

We will now show the GCD computation of a and b, and the factorization of a using the change of basis extension algorithm.

$$\begin{aligned} a &= 20\,x^{5\,m^2n-2\,n^2+n^2m^2+1} + 12\,x^{7\,m^2n-2\,n^2-m} - 30\,x^{7\,mn+n^2m^2+1} - 18\,x^{7\,mn+2\,m^2n-m} \\ b &= 8\,x^{5\,m^2n-n^2+m^2} + 12\,x^{5\,m^2n-2\,n^2+n+m} - 12\,x^{7\,mn+n^2+m^2} - 18\,x^{7\,mn+n+m} \end{aligned}$$

Example 4.4.1 Compute the GCD of a and b.

Again using Equations 4.1 – 4.3, the basis of the exponents are changed.

$$\begin{aligned} a &= 20x^{4\binom{n}{2}\binom{m}{2}+2\binom{n}{2}\binom{m}{1}+12\binom{n}{1}\binom{m}{2}+6\binom{n}{1}\binom{m}{1}-4\binom{n}{2}-2\binom{n}{1}+1} + 12x^{14\binom{n}{1}\binom{m}{2}+7\binom{n}{1}\binom{m}{1}-4\binom{n}{2}-2\binom{n}{1}-\binom{m}{1}} \\ &\quad -30x^{4\binom{n}{2}\binom{m}{2}+2\binom{n}{2}\binom{m}{1}+2\binom{n}{1}\binom{m}{2}+8\binom{n}{1}\binom{m}{1}+1} - 18x^{4\binom{n}{1}\binom{m}{2}+9\binom{n}{1}\binom{m}{1}-\binom{m}{1}} \\ b &= 8x^{10\binom{n}{1}\binom{m}{2}+5\binom{n}{1}\binom{m}{1}-2\binom{n}{2}-\binom{n}{1}+2\binom{m}{2}+\binom{m}{1}} + 12x^{10\binom{n}{1}\binom{m}{2}+5\binom{n}{1}\binom{m}{1}-4\binom{n}{2}-\binom{n}{1}+\binom{m}{1}} \\ &\quad -12x^{7\binom{n}{1}\binom{m}{1}+2\binom{n}{2}+\binom{n}{1}+2\binom{m}{2}+\binom{m}{1}} - 18x^{7\binom{n}{1}\binom{m}{1}+\binom{n}{1}+\binom{m}{1}} \end{aligned}$$

Treating n as n_1 and m as n_2, using the translation $\gamma : x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \mapsto x_{k,i_1\ldots,i_p}$ we will eliminate the exponent variables by introducing these new base variables:

$$\begin{aligned} x_{2,2} &= A = x^{\binom{n}{2}\binom{m}{2}} \\ x_{2,1} &= B = x^{\binom{n}{2}\binom{m}{1}} \\ x_{2,0} &= C = x^{\binom{n}{2}} \\ x_{1,2} &= D = x^{\binom{n}{1}\binom{m}{2}} \\ x_{1,1} &= E = x^{\binom{n}{1}\binom{m}{1}} \end{aligned}$$

$$\begin{aligned} x_{1,0} &= F &= x^{\binom{n}{1}\binom{m}{0}} \\ x_{0,2} &= G &= x^{\binom{m}{2}} \\ x_{0,1} &= H &= x^{\binom{m}{1}} \\ x_{0,0} &= I &= x \end{aligned}$$

thus producing

$$\begin{aligned} a &= 20A^4B^2C^{-4}D^{12}E^6F^{-2}I + 12C^{-4}D^{14}E^7F^{-2}H^{-1} - 30A^4B^2D^2E^8I \\ &\quad -18D^4E^9H^{-1} \qquad (4.4) \\ b &= 8C^{-2}D^{10}E^5F^{-1}G^2H + 12C^{-4}D^{10}E^5F^{-1}H \\ &\quad -12E^7C^2FG^2H - 18E^7FH \qquad (4.5) \end{aligned}$$

Standard GCD algorithms operate primarily on polynomials, disallowing negative exponents. Recall that every monomial is a unit and thus has no effect on the GCD computation when multiplied to a or b. We will multiply a and b by monomials that are large enough to remove all negative powers. This is known as the *lowest-degree unit*. Multiplying a by C^4F^2H and b by C^4F gives us

$$\begin{aligned} a_r &= 20A^4B^2D^12E^6HI + 12D^{14}E^7 - 30A^4B^2C^4D^2E^8F^2HI - \\ &\quad 18D^4E^9C^4F^2 \qquad (4.6) \\ b_r &= 8C^2D^{10}E^5G^2H + 12D^{10}E^5H - 12C^6E^7F^2G^2H - 18C^4E^7F^2H \qquad (4.7) \\ g &= \gcd(a_r, b_r) = 4D^{10}E^5 - 6E^7C^4F^2 \qquad (4.8) \end{aligned}$$

Now, substituting back to return to a symbolic polynomial with binomial basis

$$g = x^{10\binom{n}{1}\binom{m}{2}+5\binom{n}{1}\binom{m}{1}} - \frac{3}{2}x^{7\binom{n}{1}\binom{m}{1}+4\binom{n}{2}+2\binom{n}{1}}$$

Finally, using Equations 2.2 – 2.4 again, g has standard power basis for its exponent polynomials

$$g = 4x^{5nm^2} - 6x^{7nm+2n^2}$$

Example 4.4.2 Compute the factorization of a. We can use the exact same conversion techniques as seen in the previous example to transform a into the Laurent polynomial seen in Equation 4.6. The following is its factorization, in transformed, binomial basis, and standard basis:

$$\begin{aligned} f &= 2\,D^2E^6\left(2\,D^{10} - 3\,E^2C^4F^2\right)\left(3\,ED^2 + 5\,A^4B^2HI\right) \\ &= 2x^{2\binom{n}{1}\binom{m}{2}+6\binom{n}{1}\binom{m}{1}}(2x^{10\binom{n}{1}\binom{m}{2}} - 3x^{2\binom{n}{1}\binom{m}{1}+4\binom{n}{2}+2\binom{n}{1}}) \\ &\quad (3x^{\binom{n}{1}\binom{m}{1}+2\binom{n}{1}\binom{m}{2}} + 5x^{4\binom{n}{2}\binom{m}{2}+2\binom{n}{2}\binom{m}{1}+\binom{m}{1}+1}) \\ &= 2x^{nm^2+5nm}(2x^{5nm^2-5nm} - 3x^{2nm+2n^2})(3x^{nm^2} + 5x^{n^2m^2-nm^2+m+1}) \end{aligned}$$

4.5 Remarks

This section lists some important aspects of this algorithm. See Chapter 8 for an investigation of its complexity.

The GCD and factorization algorithms of Figures 4.1 and 4.2 are quite similar. This transformation algorithm is generic, working for other operations as well.

The most noticeable drawback to this algorithm is that in changing to the binomial basis we are introducing an exponential number of variables with respect to the number of symbolic exponent indeterminates. This is especially noticeable with sparse exponent polynomials. For instance, transforming $x^{m^{100}n^{100}+1} + 1$ to the binomial

exponent basis would introduce on the order of 10000 variables! The next chapter will describe a second method for operations for symbolic polynomials that does not increase the number of variables.

Chapter 5

Evaluation/Interpolation Algorithms

This chapter introduces an alternative algorithm to change of basis, using evaluation and interpolation on the exponent variables.

5.1 Projection Methods

As aforementioned, the extension algorithm of the previous chapter creates an exponential number of variables; it may be advantageous to use an evaluation/interpolation scheme if the degree of the input polynomial is large. This projection method proceeds by mapping the exponent polynomials to integers at several evaluation points, performing the desired operation, and interpolating the images of each exponent polynomial separately. Figures 5.1 and 5.2 show the dense projection algorithms for GCD and factorization respectively [Wat06].

The dense GCD projection algorithm takes as input two symbolic polynomials a, b whose symbolic exponents have maximum degree d. We assume that the evalua-

e/i_dense_gcd:
Input: $a, b \in R[n_1, \ldots, n_p; X]$
Output: $g \in R[n_1, \ldots, n_p; X]$ such that g is the GCD of a and b

1. Let d be the max degree of all exponent polynomials in a and b
2. For i from 1 to $(d+1)^p$ do
   (a) Let $e_i \in \mathbb{Z}^p$ be distinct evaluation points for $n_1, \ldots, n_p$
   (b) $g_i \leftarrow \gcd(a(e_i; X), b(e_i; X))$
3. Match corresponding terms in each g_i
4. $t \leftarrow g_1$
5. For every exponent u_1, in g_1 do
   (a) Let u_i be the matching exponent from g_i
   (b) Let j be the multivariate interpolating polynomial over the values $(e_1, u_1), \ldots, (e_{(d+1)^p}, u_{(d+1)^p})$
   (c) Replace the exponent u_1 in t with j
6. Return t

Figure 5.1: Dense Projection Algorithm for GCDs of Symbolic Polynomials

tion points are "good"; all bad specializations, which can cause linear dependencies (detailed in Section 2.4) and term collapsing (detailed in Section 5.5), are discarded. It outputs the GCD of a and b, the same result as the algorithm from Figure 4.1 up to units.

Figure 5.1 makes use of dense multivariate interpolation via the extended Vandermonde Matrix and thus requires $(d+1)^p$ evaluation points. After the initial evaluations and GCD computations, the terms of each g_i are matched. Finding corresponding terms is most easily done by matching the absolute value of the coefficients; however, if two terms share the same coefficient this can not be done. We will further discuss how to select the appropriate terms in a later section. The exponents of each

e/i_dense_factorize:
Input: $a \in R[n_1, \ldots, n_p; X]$
Output: $f_1, \ldots, f_m \in R[n_1, \ldots, n_p; X]$ such that all f_i are irreducible and their product is a

1. Let d be the max degree of all exponent polynomials in p
2. For i from 1 to $(d+1)^p$ do
   (a) Let $e_i \in \mathbb{Z}^p$ be distinct evaluation points for $n_1, \ldots, n_p$
   (b) $f_{i,1}, \ldots, f_{i,m} \leftarrow \text{factor}(a(e_i; X))$
3. Match corresponding factors of $f_{i,j}$ for varying j
4. Match corresponding terms for each $f_{i,j}$
5. $t \leftarrow f_{1,1}, \ldots, f_{1,m}$
6. For every exponent u_1, in $f_{1,1}, \ldots, f_{1,m}$ do
   (a) Let u_i be the matching exponent from $f_{i,1}, \ldots, f_{i,m}$
   (b) Let j be the multivariate interpolating polynomial over the values $(e_1, u_1), \ldots, (e_{(d+1)^p}, u_{(d+1)^p})$
   (c) Replace the exponent u_1 in t with j
7. Return t

Figure 5.2: Dense Projection Algorithm for Factorization of Symbolic Polynomials

term of each GCD are then interpolated, monomial by monomial, and reconstructed to form the desired result.

The multivariate interpolation step is essentially solving a system of linear equations; we must be certain that the Vandermonde system is nonsingular. Dependent evaluation points must be discarded. It is also possible to find several g_i with a varying number of terms. This is due to two or more terms collapsing together. These smaller GCDs, with fewer terms should be discarded as well. If a GCD is found with more terms, all previous images contain a collapse of terms and are discarded.

The factorization algorithm is nearly identical to its GCD counterpart, save for the fact that it has one input polynomial and outputs several polynomials. As such, it suffers from the same term identification problems as the GCD algorithm. This is in addition to the classical problem of matching corresponding factors.

5.2 Dense Projection Examples

We will now show the GCD computation of a and b and the factorization of a using the dense projection methods of the previous section.

$$\begin{aligned} a &= 14\,y^{7-10\,m^2-18\,mn-16\,n^2}x^{-13\,m^2-21\,n+19+2\,mn-5\,n^2} + \\ &\quad 4\,y^{19+8\,m+13\,m^2-10\,mn}x^{-23\,m^2-2\,n-10+16\,m-17\,mn} - \\ &\quad 63\,y^{-16\,mn+7-10\,m^2-16\,n^2}x^{19-19\,n+2\,mn-5\,n^2} - \\ &\quad 18\,y^{-8\,mn+19+8\,m+13\,m^2}x^{-10+16\,m-10\,m^2-17\,mn} \qquad (5.1) \\ b &= 8\,y^{-18\,m-4\,n-20\,mn-16\,n^2}x^{-13\,m^2+9\,n+3-5\,m-10\,mn} - \\ &\quad 10\,y^{-18\,m+18\,n-19\,m^2-9\,mn}x^{-13\,m^2+9\,n-1+9\,mn+10\,n^2} - 2\,x^{-19\,m^2+12\,n+5\,m+5\,n^2} - \\ &\quad 36\,y^{-18\,mn-18\,m-4\,n-16\,n^2}x^{3-5\,m+11\,n-10\,mn} + \\ &\quad 45\,y^{-7\,mn-18\,m+18\,n-19\,m^2}x^{-1+11\,n+9\,mn+10\,n^2} + 9\,y^{2\,mn}x^{5\,m+14\,n-6\,m^2+5\,n^2} \qquad (5.2) \end{aligned}$$

Example 5.2.1 Compute the GCD of a and b. The exponents of the two input polynomials are bivariate and have a maximum degree of 2, thus requiring $(2+1)^2 = 9$ evaluation points to interpolate. The nine randomly selected evaluation points, and their respective GCD can be found in Table 5.1.

The final GCD polynomial will be of the form $-2+9y^{p_1}x^{p_2}$, where p_1 and p_2 are of the form $c_8n^2m^2+c_7n^2m+c_6n^2+c_5nm^2+c_4nm+c_3n+c_2m^2+c_1m+c_0$. Two multivari-

n	m	$\gcd(a(n,m;x,y),b(n,m;x,y))$
3	5	$-2+9y^{30}x^{331}$
7	4	$-2+9y^{56}x^{222}$
2	6	$-2+9y^{24}x^{472}$
4	3	$-2+9y^{24}x^{125}$
9	4	$-2+9y^{72}x^{226}$
9	7	$-2+9y^{126}x^{655}$
2	4	$-2+9y^{16}x^{212}$
8	8	$-2+9y^{128}x^{848}$
5	8	$-2+9y^{80}x^{842}$

Table 5.1: Evaluation Points for Example 5.2.1.

ate interpolations must be performed, one to solve for the coefficients in p_1, the other for p_2. Interpolating the evaluation points over $[30, 56, 24, 24, 72, 126, 16, 128, 80]$, we find $p_1 = 2mn$. Interpolating over $[331, 222, 472, 125, 226, 655, 212, 848, 842]$, we find $p_2 = 2n + 13m^2$. Thus, the final output GCD symbolic polynomial is

$$-2 + 9y^{2mn}x^{2n+13m^2}$$

Example 5.2.2 Compute the factorization of a. As with the previous example, nine (n, m) pairs are required for interpolation. Using the same values as the previous example, the factors of each evaluation can be found in Table 5.2.

n	m	$factor(a(n,m;x,y))$
3	5	$(-9y^{30}x^{331}+2)(7x^{382}+2y^{891})$
7	4	$(-9y^{56}x^{222}+2)(7x^{279}+2y^{1420})$
2	6	$(-9y^{24}x^{472}+2)(7x^{463}+2y^{1048})$
4	3	$(-9y^{24}x^{125}+2)(7x^{143}+2y^{595})$
9	4	$(-9y^{72}x^{226}+2)(7x^{233}+2y^{1996})$
9	7	$(-9y^{126}x^{655}+2)(7x^{1028}+2y^{2995})$
2	4	$(-9y^{16}x^{212}+2)(7x^{219}+2y^{540})$
8	8	$(-9y^{128}x^{848}+2)(7x^{1285}+2y^{3084})$
5	8	$(-9y^{80}x^{842}+2)(7x^{1081}+2y^{2268})$

Table 5.2: Evaluation Points for Example 5.2.2.

There will be two output factors, which will be of the form $-9y^{p_1}x^{p_2}+2$ and $7x^{p_3}+2y^{p_4}$, where $p_1, p_2, p_3, p_4 = c_8n^2m^2+c_7n^2m+c_6n^2+c_5nm^2+c_4nm+c_3n+c_2m^2+c_1m+c_0$. Four multivariate interpolations are to be done, one for each p_i. Interpolating the n and m pairs over $[30, 56, 24, 24, 72, 126, 16, 128, 80]$, we find $p_1 = 2nm$. Likewise, interpolating the exponents of the remaining p_i, we find

$$p_1 = 2mn \tag{5.3}$$

$$p_2 = 13m^2 + 2n \tag{5.4}$$

$$p_3 = 10m^2 + 19mn - 16m - 5n^2 - 19n + 29 \tag{5.5}$$

$$p_4 = 23m^2 + 8mn + 8m + 16n^2 + 12 \tag{5.6}$$

Thus, the final output factors of a are

$$(-9y^{2mn}x^{13m^2+2n} + 2)(7x^{10m^2+19mn-16m-5n^2-19n+29} + 2y^{23m^2+8mn+8m+16n^2+12}) \tag{5.7}$$

5.3 Sparse Interpolation

As previously mentioned, using the naive approach of the previous section requires on the order of $(d+1)^p$ evaluations. In practice, symbolic polynomials are sparse. Using sparse interpolation instead of dense, the algorithm will no longer require an exponential number of evaluation points. For *t-sparse* symbolic polynomials, we can reduce the minimum number of required evaluations to $O(pdt)$. The sparse projection GCD algorithm can be seen in Figure 5.3. This algorithm incorporates notions from its dense relative (Figure 5.1), and Zippel interpolation (Figure 2.2). The algorithm takes as input symbolic polynomials a and b.

As output, the algorithm produces the GCD of a and b, the exact result of pre-

e/i_sparse_gcd:
Input: $a, b \in R[n_1, \ldots, n_p; X]$
Output: $g \in R[n_1, \ldots, n_p; X]$ such that g is the GCD of a and b

1. Let d be the max degree of all exponent polynomials in a and b
2. Let $e_{i,j}$ be the i^{th} evaluation for n_j
3. $E_1 \leftarrow e_{1,i}, e_{2,1}, \ldots, e_{v-1,1}, e_{v,1}$
4. $r_1 \leftarrow \gcd(a(E_1; X), b(E_1; X))$
5. $t \leftarrow 1$
6. For i from 1 to p do
   (a) For j from 2 to $d + 1$ do
      i. For k from 1 to t do
         A. $E_k \leftarrow e_{1,k}, \ldots, e_{i-1,k}, e_{i,j}, e_{i+1,1}, \ldots, e_{v,1}$
         B. $g_k \leftarrow \gcd(a(E_k; X), b(E_k; X))$
      ii. $r_j \leftarrow g_1$
      iii. For every exponent u_1, in g_1 do
         A. Let u_k be the matching exponent from g_k
         B. Let q be the matching exponent from r_1
         C. Using u_k and E_k, let s be the polynomial with the form of q whose coefficients are the solution of the system of t linear equations
         D. Replace the exponent u_1 in r_j with s
   (b) Let $r_1 \in R[n_1, \ldots, n_i; X]$ be the symbolic polynomial obtained by interpolating matching exponent values in r_j over $e_{j,1}, \ldots, e_{j,d+1}$
   (c) Let t be the max number of terms of all exponent polynomials in r_1
7. Return r_1

Figure 5.3: Sparse Projection Algorithm for GCDs of Symbolic Polynomials

viously described symbolic polynomial GCD methods up to units. We once again make the assumption of having only "good" evaluation points. When the algorithm commences, there is no knowledge of the form of the output polynomial. A dense interpolation for n_1 is performed, by varying n_1, $d+1$ times, and keeping the other exponent variables constant. After this initial interpolation, we can be certain of the number of exponent polynomials, the base coefficients and the base variables in each term. We are now basically performing classical Zippel interpolation for every exponent polynomial using the same evaluation points.

As this is a revision of the dense version, some of its inherent problems may arise, namely term identification of two monomials with identical coefficients and term collapsing. Moreover, it is possible that evaluations produce dependent equations, which must be removed to ensure a consistent linear system. If too many equations are thrown away, while unlikely, it is possible that there will not be a sufficient number of remaining evaluations, and a new set of points must be used. This is because individual values are shared across several evaluations. This problem is specific only to the sparse algorithm and does not arise in dense interpolation.

As with all of the preceding factorization algorithms, Figure 5.4 is very similar to the sparse GCD algorithm, the only difference being that factorization can return a set of polynomials rather than just a single one.

5.4 Sparse Projection Examples

We will now show the GCD computation of a (Equation 5.1) and b (Equation 5.2) and discuss the factorization of a using the sparse projection methods of the previous section.

e/i_sparse_factorize:
Input: $a \in R[n_1, \ldots, n_p; X]$
Output: $f_1, \ldots, f_m \in R[n_1, \ldots, n_p; X]$ such that all f_i are irreducible and their product is a

1. Let d be the max degree of all exponent polynomials in p and q
2. Let $e_{i,j}$ be the i^{th} evaluation for n_j
3. $E_1 \leftarrow e_{1,i}, e_{2,1}, \ldots, e_{v-1,1}, e_{v,1}$
4. $r_{1,1}, \ldots, r_{1,m} \leftarrow \text{factor}(a(E_1; X))$
5. $t \leftarrow 1$
6. For i from 1 to p do
   (a) For j from 2 to $d+1$ do
      i. For k from 1 to t do
         A. $E_k \leftarrow e_{1,k}, \ldots, e_{i-1,k}, e_{i,j}, e_{i+1,1}, \ldots, e_{v,1}$
         B. $f_{k,1}, \ldots, f_{k,m} \leftarrow \text{factor}(a(E_k; X))$
      ii. $r_{j,1}, \ldots, r_{j,m} \leftarrow f_{1,1}, \ldots, f_{1,m}$
      iii. For every exponent u_1, in $f_{1,1}, \ldots, f_{1,m}$ do
         A. Let u_k be the matching exponent from $f_{k,1}, \ldots, f_{k,m}$
         B. Let b be the matching exponent from $r_{1,1}, \ldots, r_{1,m}$
         C. Using u_k and E_k, let s be the polynomial with the form of b whose coefficients are the solution of the system of t linear equations
         D. Replace the exponent u_1 in $r_{j,1}, \ldots, r_{j,m}$ with s
   (b) Let $r_{1,1}, \ldots, r_{1,m} \in R[n_1, \ldots, n_i; X]$ be the symbolic polynomials obtained by interpolating matching factors and exponent values in $r_{j,k}$ over $e_{j,1}, \ldots, e_{j,d+1}$
   (c) Let t be the max number of terms of all exponent polynomials in $r_{1,1}, \ldots, r_{1,m}$
7. Return $r_{1,1}, \ldots, r_{1,m}$

Figure 5.4: Sparse Projection Algorithm for Factorization of Symbolic Polynomials

Example 5.4.1 Compute the GCD of a and b. The exponents of the two input polynomials are bivariate and have a maximum degree of two, thus each variable requires at most three values. This gives a total of nine possible evaluation points; however, they will not all be needed. For simplicity, we will assume we know the sparsity of the resulting polynomials; it has two terms. The seven required evaluation points and their respective GCDs are found in Table 5.4.1.

n	m	$\gcd(a(n,m;x,y), b(n,m;x,y))$
3	3	$-2+9y^{18}x^{123}$
3	4	$-2+9y^{24}x^{214}$
3	5	$-2+9y^{30}x^{331}$
4	3	$-2+9y^{24}x^{125}$
4	4	$-2+9y^{32}x^{216}$
5	3	$-2+9y^{30}x^{127}$
5	4	$-2+9y^{40}x^{218}$

Table 5.3: Evaluation Points for Example 5.4.1.

By keeping n constant at $n = 3$, we can use the first three evaluations to interpolate each symbolic exponent for m. Interpolating $[18, 24, 30]$ and $[123, 214, 331]$ over the m values $[3, 4, 5]$ gives us the polynomials $p_1 = 6m$ and $p_2 = 13m^2 + 6$ respectively. According to Zippel, it is highly improbable that p_1 will contain an m^2 or a constant term, and that p_2 will contain an m monomial. At this point we are certain that our final GCD will be of the form $-2 + 9x^{Am^2+B}y^{Cm}$, where $A, B, C \in \mathbb{Z}[n]$. We call $a_1 = Am^2 + B$ and $a_2 = Cm$ our anchors. Substituting p_1 and p_2 into their respective anchors gives us

$$g_1 = -2 + 9x^{13m^2+6}y^{6m} \tag{5.8}$$

Two more symbolic polynomials of the above form are needed to interpolate for n; however, as the exponents have at most two terms, constructing these polynomials only requires two evaluations each. By using the $n = 4$ evaluations and substituting

into the anchors we obtain a systems of linear equations for a_1,

$$9A + B = 125 \tag{5.9}$$

$$16A + B = 216 \tag{5.10}$$

which when solved yields $A = 13$ and $B = 8$. Substituting into the second anchor the following system of equations is constructed,

$$3C = 24 \tag{5.11}$$

$$4C = 32 \tag{5.12}$$

which is trivially solved for $C = 8$. After substituting A, B, C into the anchors we get the second polynomial used to interpolate for n

$$g_2 = -2 + 9x^{13m^2+8}y^{8m} \tag{5.13}$$

The exact steps are repeated, but for the $n = 5$ evaluations. The first system of equations is,

$$9A + B = 127 \tag{5.14}$$

$$16A + B = 218 \tag{5.15}$$

which holds true for $A = 13, B = 10$. The second system,

$$3C = 30 \tag{5.16}$$

$$4C = 40 \tag{5.17}$$

is trivially $C = 10$. After substituting again, the final intermediate GCD is obtained

$$g_3 = -2 + 9x^{13m^2+10}y^{10m} \tag{5.18}$$

Now, the matching exponent polynomial coefficients in g_i are interpolated independently. This results in the goal exponent polynomials in $\mathbb{Z}[n, m]$. Interpolating for A in a_1 is trivial, as it is 13 in all g_i. Interpolating $[6, 8, 10]$ over the n values $[3, 4, 5]$ results in $2n$, thus $B = C = 2n$. Substituting these polynomials in for A, B, C will give us the end GCD of

$$-2 + 9x^{13m^2+2n}y^{2nm} \tag{5.19}$$

Recall that the dense version of solving this exact problem required nine GCD evaluations. Using sparse interpolation saved two evaluation computations. In this low degree example, the factorization computation of a using sparse interpolation would proceed exactly the same as the previous example. Note that the factorization of a (as already calculated in Example 5.2.2) has a maximum number of exponential terms of three. As its degree is only two, the exponent polynomials are not sparse enough to take advantage of Zippel's algorithm.

5.5 Bad Evaluation Points and Term Selection

Consider interpolating for the symbolic polynomial p using the given evaluation images:

$$p(3; x) = 7 + 11x \tag{5.20}$$

$$p(4; x) = 18 \tag{5.21}$$

$$p(5; x) = 7x + 11 \tag{5.22}$$

At first glance, it is not clear how to match the exponents to find the solution, $p = 7x^{m-4} - 11$. We can see that the symbolic polynomial evaluated at $n = 4$ is a constant, while evaluated at 3 and 5, there are two terms; however, the base variable x is attached to different coefficients. In this form, interpolation cannot be done on these images.

Equation 5.21 is smaller than the other evaluations due to collapsing terms. There is always a possibility that two or more exponents will evaluate to the same value, in which case, when simplified, the like terms will be added together. In this example, the exponent of x maps to zero, and $p(4; x)$ becomes the sum 7 and 11. It may be possible to perform some reverse engineering to attempt to "unsimplify" the results, but it is easier to disregard the evaluation and obtain a new image. Term collapsing always lowers the number of terms in the evaluated image. If at some point a new image with more terms is evaluated, the preceding images must be removed as they all contained a collapse of terms. For GCDs of a single exponent variable, this can occur for at most $dt(t-1)/2$ points, with t the number of exponent polynomials and d the maximum degree of those polynomials [Wat07a]. There may be an infinite number of the bad evaluation points for multivariate exponents; however, by randomly choosing evaluation points, their probability of arising is very small.

Equations 5.20 and 5.22 still cannot be interpolated, due to variable placement. Recall that these are Laurent polynomials, and as such, multiplying by any monomial will leave a GCD or factorization unchanged. We can always normalize by multiplying our evaluated images by monomials to make one of the terms constant. This will ensure that the variables of the remaining terms match. Multiplying Equation 5.22 by x^{-1}, we have

$$p(5; x) = 7 + 11x^{-1} \tag{5.23}$$

which then matches the form of Equation 5.20 and they can be used together in an interpolation.

Term identification of symbolic polynomials with identical absolute coefficients can also be problematic. Consider the interpolation of a symbolic polynomial, p, with one exponent variable and maximum degree of 3, using the following images

$$p(2;x) = 5x^{18} + 5x^{9} + 3x^{-10} \tag{5.24}$$

$$p(4;x) = 5x^{65} + 5x^{42} + 3x^{-20} \tag{5.25}$$

$$p(5;x) = 5x^{126} + 5x^{60} + 3x^{-25} \tag{5.26}$$

$$p(6;x) = 5x^{217} + 5x^{82} + 3x^{-30} \tag{5.27}$$

We wish to independently interpolate the exponents of each term, but in this case we have two terms with the coefficient of 5. We have no direct means of knowing which matching exponent values to select when interpolating. Watt [Wat07a], has shown a few different methods of grouping corresponding terms which we describe in the following three sections.

5.5.1 Brute Force

The first idea involves trying all possible correspondences until the correct one is found. Using four evaluations points, every permutation will be interpolated by a polynomial of degree 3 or less. However, if we introduce a fifth point, the only interpolating polynomial of degree 3 or less is the desired ordering of properly matched terms. We introduce the following image.

$$p(7;x) = 5x^{344} + 5x^{108} + 3x^{-35} \tag{5.28}$$

We have two confounded terms, and 5 evaluations points, giving a total of $2^4 = 16$ possible assignments. Table 5.5 shows all possible correspondences. There is a column for each term, and a row for each correspondence. Each entry in the table contains a list of exponents for that term for each evaluation. Note that the final column is constant; term identification is not needed as its exponents are known.

Corres.	Term 1 $5x^{e_1}$	Term 2 $5x^{e_2}$	Term 3 $3x^{e_3}$
1	[18,65,126,217,344]	[9,42,60,82,108]	[-10,-20,-25,-30,-35]
2	[18,65,126,217,108]	[9,42,60,82,344]	[-10,-20,-25,-30,-35]
3	[18,65,126,82,344]	[9,42,60,217,108]	[-10,-20,-25,-30,-35]
4	[18,65,126,82,108]	[9,42,60,217,344]	[-10,-20,-25,-30,-35]
5	[18,65,60,217,344]	[9,42,126,82,108]	[-10,-20,-25,-30,-35]
6	[18,65,60,217,108]	[9,42,126,82,344]	[-10,-20,-25,-30,-35]
7	[18,65,60,82,344]	[9,42,126,217,108]	[-10,-20,-25,-30,-35]
8	[18,65,60,82,108]	[9,42,126,217,344]	[-10,-20,-25,-30,-35]
9	[18,42,126,217,344]	[9,65,60,82,108]	[-10,-20,-25,-30,-35]
10	[18,42,126,217,108]	[9,65,60,82,344]	[-10,-20,-25,-30,-35]
11	[18,42,126,82,344]	[9,65,60,217,108]	[-10,-20,-25,-30,-35]
12	[18,42,126,82,108]	[9,65,60,217,344]	[-10,-20,-25,-30,-35]
13	[18,42,60,217,344]	[9,65,126,82,108]	[-10,-20,-25,-30,-35]
14	[18,42,60,217,108]	[9,65,126,82,344]	[-10,-20,-25,-30,-35]
15	[18,42,60,82,344]	[9,65,126,217,108]	[-10,-20,-25,-30,-35]
16	[18,42,60,82,108]	[9,65,126,217,344]	[-10,-20,-25,-30,-35]

Table 5.4: Correspondences for Term Identification.

The final correspondence is the only one which interpolates polynomials with degrees that are not 4. We can be certain that this is the correct matching of terms. Using these interpolating polynomials, we can construct the desired answer of

$$p(n; x) = 5x^{2n^2+10} + 5x^{n^3+1} + 3x^{-5n} \tag{5.29}$$

With t confounded symbolic exponents and n evaluation points, there are a total of $t!^{n-1}$ possible correspondences that must be tested. This doubly exponential com-

plexity is undesirable, and only feasible for very small values of n and t.

5.5.2 Ordering at Extreme Values

An alternative approach corresponds terms by observing that when choosing evaluation points that are "large enough", the corresponding terms of the image will have a consistent order according to degree. For univariate exponents, the exponent polynomials intersect a finite number of times. If we were to choose evaluations larger than the largest intersection point, then we are ensured that the resulting evaluations will be well ordered. For instance, all images of Equation 5.29 evaluated at $n > 3$ are well ordered. This method of selecting corresponding terms is not viable in practice. Firstly, we do not know the interpolating polynomial, and are unsure of how large "large enough" really is. Moreover, by choosing larger evaluation points, the degree of the images grows dramatically, increasing the computation time of the image GCDs.

5.5.3 Interpolation of Symmetric Functions

The final, and best suited method of term correspondence, is by performing the interpolation of symmetric functions. The *Elementary Symmetric Functions*, π_k, over n variables $\{x_1, \ldots, x_n\}$ are defined as

$$\pi_1(x_1, \ldots, x_n) = \sum_{1 \leq i \leq n} x_i \tag{5.30}$$

$$\pi_2(x_1, \ldots, x_n) = \sum_{1 \leq i \leq j \leq n} x_i x_j \tag{5.31}$$

$$\vdots$$

$$\pi_n(x_1, \ldots, x_n) = \prod_{1 \leq n} x_i \tag{5.32}$$

For example, the elementary symmetric functions for three variables are:

$$\pi_1(x_1, x_2, x_3) = x_1 + x_2 + x_3 \tag{5.33}$$

$$\pi_2(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3 \tag{5.34}$$

$$\pi_3(x_1, x_2, x_3) = x_1x_2x_3 \tag{5.35}$$

If we have s terms that are indistinguishable, $t_1, \ldots, t_s$, we can interpolate $\pi_j(t_1, \ldots, t_s)$ over different values of j, and then solve for some t_i by using an evaluation to break the symmetry. We will once again attempt to interpolate for p, using the evaluations from Equations 5.24 – 5.28. There are two indistinguishable terms, say $A(n)$ and $B(n)$, each of degree 3 of the form

$$A(n) = a_3n^3 + a_2n^2 + a_1n + a_0 \tag{5.36}$$

$$B(n) = b_3n^3 + b_2n^2 + b_1n + b_0 \tag{5.37}$$

and thus we require the two variable elementary symmetric functions

$$\pi_1(A, B) = A(n) + B(n) \tag{5.38}$$

$$\pi_2(A, B) = A(n) \times B(n) \tag{5.39}$$

If we substitute the above equations for A and B we get.

$$A(n) + B(n) = (a_3 + b_3)n^3 + (a_2 + b_2)n^2 + (a_1 + b_1)n + a_0 + b_0 \tag{5.40}$$

$$\begin{aligned} A(n) \times B(n) = {} & (a_3b_3)n^6 + (a_2b_3 + a_3b_2)n^5 + (a_1b_3 + a_2b_2 + a_3b_1)n^4 + \\ & (a_0b_3 + a_1b_1 + a_2b_1 + a_3b_0)n^3 + (a_0b_2 + a_1b_1 + a_2b_0)n^2 + \\ & (a_0b_1 + a_1b_0)n + a_0b_0 \end{aligned} \tag{5.41}$$

We wish to interpolate these two functions. The degree of $\pi_2(A, B)$ is at most 6. An additional two evaluations are required.

$$p(8; x) = 5x^{513} + 5x^{138} + 3x^{-40} \quad (5.42)$$

$$p(9; x) = 5x^{730} + 5x^{172} + 3x^{-45} \quad (5.43)$$

The following table details the calculation of the elementary symmetric polynomials and their interpolation.

n	$A(n) + B(n)$	$A(n) \times B(n)$
2	$18 + 9 = 27$	$18 \times 9 = 162$
4	$65 + 42 = 107$	$65 \times 42 = 2730$
5	$126 + 60 = 186$	$126 \times 60 = 7560$
6	$217 + 82 = 299$	$217 \times 82 = 17794$
7		$344 \times 108 = 37152$
8		$513 \times 138 = 70794$
9		$730 \times 172 = 125560$
Interp	$n^3 + 2n^2 + 11$	$2n^5 + 10n^3 + 2n^2 + 10$

Table 5.5: Correspondences for Term Identification.

The coefficients of the interpolating symmetric polynomials are set equal to their respective coefficient equations from Equations 5.40 and 5.41. This creates a system of 11 equations with 8 unknowns; however, this has an infinite number of solutions. The symmetry must be broken by assigning n an arbitrary value, say $n = 2$. Thus, we have an additional two equations

$$a_3 + a_2 + a_1 + a_0 = 18 \quad (5.44)$$

$$b_3 + b_2 + b_1 + b_0 = 9 \quad (5.45)$$

and a system with a single solution, namely

$$a_3 = 0 \quad a_2 = 2 \quad a_1 = 0 \quad a_0 = 0$$
$$b_3 = 1 \quad b_2 = 0 \quad b_1 = 0 \quad b_0 = 1$$

This gives us the final interpolated symbolic polynomial of

$$p(n; x) = 5x^{2n^2} + 5x^{n^3+1} + 3x^{-5n} \tag{5.46}$$

If we have s indistinguishable terms, of max degree d, this method requires $O(ds)$ evaluations to match corresponding terms. [Wat07a]

5.6 Remarks

At first glance, sparse interpolation would seem to be the method of choice – dense evaluation/interpolation requires an exponential number of images and extension methods create an exponential number of variables. However, the degree bound of the evaluated exponent polynomials is often too large (in the millions) for computer algebra systems to compute GCDs or factorizations in a timely manner. In practice, evaluations are chosen as small as possible to hopefully minimize the degree of the images, but this is insufficient. The degrees are still huge.

Interpolation might have the potential to outperform the change of basis method. The next chapter investigates more intelligent methods of selecting evaluation points that will result in images of smaller degree.

Chapter 6

Better Evaluation Points

The sparse interpolation of the exponent polynomials allows for a linear number of evaluations; however, we are still left with the problem of the exponent polynomials evaluating to large values. The degree of the evaluated images has a huge influence over the computation time, and is determined solely by the selection of values for the exponent polynomials. In this chapter we will introduce two methods of selecting evaluation points for symbolic polynomials that will lower the degree bound of the evaluated images.

6.1 Optimizing Point Selection

The evaluated Laurent Polynomial images are normalized by dividing by the lowest degree unit before the GCD or factorization computation. As such, we are not looking for the values that will give us the lowest degree of the exponent polynomials, but rather the values that give the minimal difference between the largest and smallest evaluated degree. Exponent polynomials are integer-valued; our output points should be integers as well.

This idea is best illustrated through the following example. Consider the symbolic polynomial

$$a(n;x) = a_1 x^{p(n)} + a_0 x^{q(n)} \tag{6.1}$$

We want to choose values of n such that $|p(n) - q(n)|$ is as small as possible. If there was a single exponent polynomial, the values surrounding its roots and local minima would be selected.

This notion is shown in Figure 6.1. We have plotted a polynomial, $p(n)$, of degree 4. The three smallest integer-valued points that lie on $p(n)$, are $n = 1, 2, 3$, all near its root. However, if we were to interpolate $p(n)$, we would need 5 evaluations. The five smallest points near the root of $p(n)$ are shown as a square. The five smallest points near the root or any local minima are shown as a large circle. A better solution can be found by using the points near the local minima.

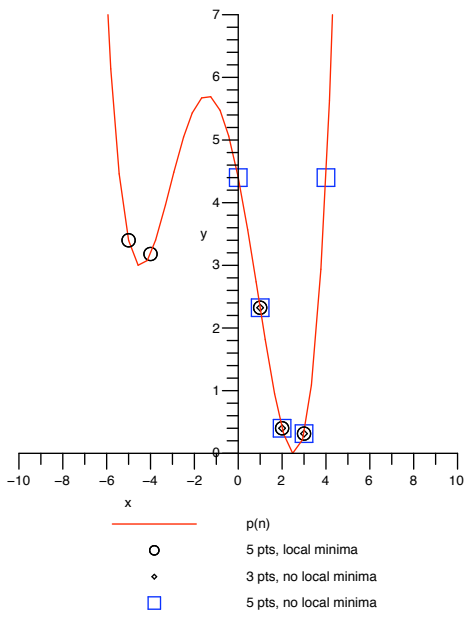
The complexity of the problem grows by allowing for an arbitrary number of exponent polynomials with an arbitrary number of indeterminates. Figure 6.2 shows several exponent polynomials, $f_i(n)$. The vertical distance between the largest and smallest polynomials for all of n is shaded grey. We are looking for the integer values of n that minimize that vertical distance.

6.1.1 Problem Statement

The problem of finding k points that minimize the maximal difference of the r input exponent polynomials can be generalized to the following:

Given r polynomials $p_1, p_2, \ldots, p_r \in \mathbb{Q}[X_1, X_2, \ldots, X_v]$, find k distinct values $n_1, n_2, \ldots, n_k \in \mathbb{Z}$ that provide the k smallest values for $P_M - P_m = \max\{p_1(n), p_2(n), \ldots, p_r(n)\} - \min\{p_1(n), p_2(n), \ldots, p_r(n)\}$.

A simple approximate solution to this problem would be to evaluate at many



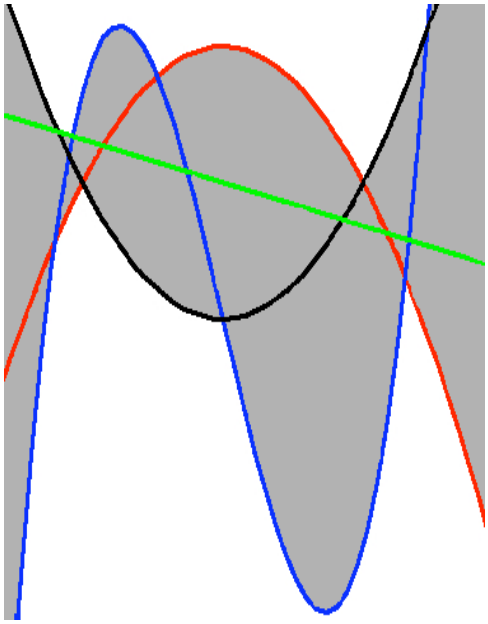


Figure 6.1: Evaluation Point Selection for one Exponent Polynomial (Left)

Figure 6.2: Several Exponent Polynomials (Right)

points, selecting the ones which gives the smallest $P_M - P_m$. This does not guarantee the best solutions. For multivariate problems, an exponential number of sample points, with respect to the degree and number of variables, would need to be taken to get a proper representation of the problem space.

6.1.2 A Symbolic Solution

An alternative solution is to solve for the two piecewise polynomials P_M and P_m, and then find the k points that minimize their difference. The optimal points will be found surrounding the local minima and noncontinuous points of $P_M - P_m$. This guarantees to produce the optimal output values. For the univariate case, a maximum of $r!$ intersections must be solved for this construction.

This symbolic procedure is best suited when given a reasonable number of uni-

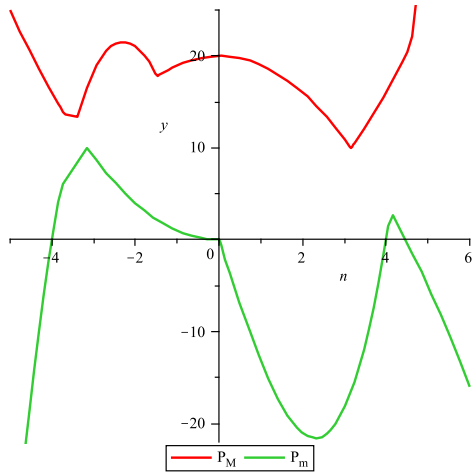


Figure 6.3: Piecewise Max and Min Polynomials for Example 6.1.1 (Left)

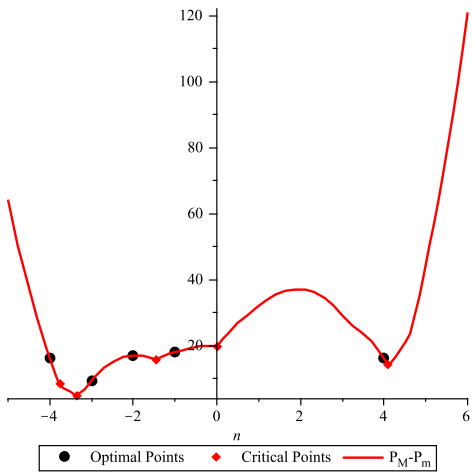


Figure 6.4: The Optimal Integer Points of $P_M - P_m$ (Right)

variate polynomials. Finding P_M and P_m for multivariate input is infeasible.

Example 6.1.1 Find the 5 points that minimize $P_M - P_m$ for the polynomials in Figure 6.2

The two piecewise polynomials of Figure 6.3 are built by first finding the largest (smallest) polynomial at $-\infty$, p_1. Moving from $-\infty$ to ∞ we will find all the intersections of P_M (P_m). The remaining polynomials are tested to see if they intersect with p_1. The closest intersection with p_1 is the new largest (smallest) polynomial for the piecewise function. This process is continued until the topmost (bottommost) polynomial has no more further positive intersections available.

We can take their difference to find $P_M - P_m$, as seen in Figure 6.4. We know the equations of every piece of $P_M - P_m$, as well as its non-continuous points of intersection. Like the example from Figure 6.1, the optimal values are going to be surrounding all of these critical points. The five integer points that minimize $P_M - P_m$ are seen as black dots in Figure 6.4.

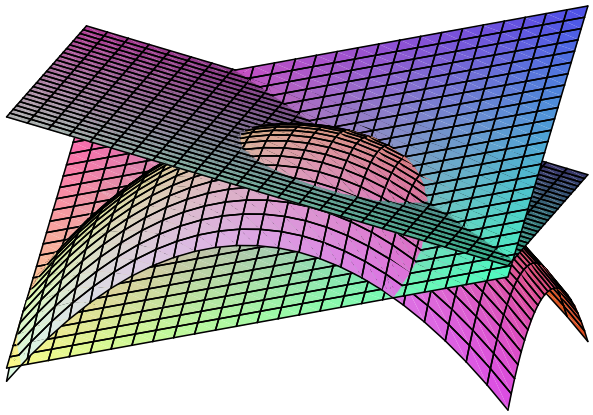


Figure 6.5: Several Multivariate Exponent Polynomials

6.1.3 A Numerical Solution

The above symbolic method is much less straightforward, and more computationally expensive when the input is multivariate. Constructing the piecewise polynomials P_M and P_m for a small example, as seen in Figure 6.5 is hard, however, it is easy to numerically evaluate $P_M - P_m$ at values. A numerical heuristic which attempts to minimize this difference is a much more suitable choice for this task.

The optimal points may be found on, or near the intersection of a certain number of polynomials. We propose a method that performs steepest descent on a polynomial until it intersects with another. It then follows that intersection curve until it meets with a third polynomial. It then follows the intersection of all three polynomials. This continues onward, and finishes at an intersection of every polynomial or a local minima. The algorithm is shown below as Figure 6.6.

We want to find several local minima so the main procedure is iterated m times. The algorithm begins by choosing a random point l. All polynomials are evaluated

Input: $\{p_1, p_2, \ldots, p_r\}, p_i \in \mathbb{Q}[X_1, X_2, \ldots, X_v]$
: $k \in \mathbb{Z}$
Output: $\{o_1, o_2, \ldots, o_k\}, o_i \in \mathbb{Z}^v$

1. Let m be the number of rounds to perform
2. Let n be the maximum number of descent iterations
3. Let $ord : (\{\mathbb{Q}[X_1, \ldots, X_v]^r\}, \mathbb{Z}^v) \rightarrow (\mathbb{Q}[X_1, \ldots, X_v]^r)$ be a function that sorts its input polynomials in descending order according to their evaluation of the second argument
4. Let $f : (\{\mathbb{Q}[X_1, \ldots, X_v]^r\}, \mathbb{Z}^v) \rightarrow \mathbb{Z}$, such that $f(\{q_1, \ldots, q_v\}, e) = max(q_1(e), \ldots, q_v(e)) - min(q_1(e), \ldots, q_v(e))$.
5. For i from 1 to m do
   (a) Select a random point $l \in \mathbb{Z}^n$
   (b) $S \leftarrow ord(\{p_1, \ldots, p_r\}, l)$
   (c) $c \leftarrow \{\}$
   (d) $t \leftarrow S_1 \quad b \leftarrow S_r \quad g \leftarrow l$
   (e) For j from 1 to n do
      i. Let g be the minimal value obtained by steepest descent on $S_1 - S_r$ at the point g, subject to constraints c and recalculating S at each step. If S_1 or S_r change the descent halts.
      ii. If g is a local minimum then break
      iii. If $t \neq S_1$ then
         A. $c \leftarrow c \cup \{t - S_1\}$
         B. $t \leftarrow S_1$
      iv. If $b \neq S_r$ then
         A. $c \leftarrow c \cup \{b - S_r\}$
         B. $t \leftarrow S_r$
   (f) Take the k integer points in every direction of $round(g)$ and add them to $pset$
6. Return the k elements in $pset$ that minimize f

Figure 6.6: Numerical Steepest Descent Algorithm for Evaluation Point Selection for Symbolic Polynomials

and sorted at l. The objective function $S_1 - S_r$ (the numerical equivalent of $P_M - P_m$) is selected and follows the path of steepest descent until there is a new topmost (when $t \neq S_1$) or bottommost (when $b \neq S_r$) polynomial, or a local minima is found. In the latter case the search has completed, for the former we want to minimize the same objective function while remaining on the topmost and bottommost curves of intersection. This is done by setting the intersection curves as constraints. As new pieces of P_M and P_m are found additional constraints are added to the minimization.

Testing of the numerical method has shown considerable runtime improvement over its symbolic counterpart. This is a heuristic. If the intersection of all polynomials is not found, it does not guarantee the best solution.

6.2 Evaluating at Primitive Roots of Unity

Some background information on roots of unity can be found in Section 2.7. We will take P as the next prime larger than the maximum degree of the exponent polynomials. The initial motivation behind evaluating at these special complex, irrational values was that the powers of ω_P can not exceed $P - 1$ from its periodic properties. The effectiveness of the point optimization algorithm of the last section depends greatly on the set of exponent polynomials. Evaluating at primitive roots of unity provides a consistent alternative.

The algorithm will use ideas from the extension and projection methods. Each exponent variable is first independently evaluated at a primitive P^{th} root of unity with the powers of ω reduced modulo P. The exponent polynomials are now univariate in ω with highest possible degree of $P - 1$. Treating ω as a variable we use the algebraic independence of $x^{\omega}, x^{\omega^2}, \ldots$ to introduce new variables similar to the extension method of Chapter 4; likewise, we must account for fixed divisors. Each

exponent polynomial is then mapped to the binomial basis. As the intermediate ω polynomials are univariate, no additional variables are introduced in this translation. The reduction of variables comes at a cost – the degree is no longer bounded by P, it grows exponentially. Variables are then substituted for each $x_i^{\omega^j}$. This results in a Laurent polynomial.

We will now demonstrate the above idea through example.

Example 6.2.1 Evaluate a at $(n, m) = (\omega^2, \omega)$ and map to a Laurent polynomial, factor and map the exponents back to the power basis.

$$a(n,m;x) = 20\,x^{5\,m^2n-2\,n^2+n^2m^2+1}+12\,x^{7\,m^2n-2\,n^2-m}-30\,x^{7\,mn+n^2m^2+1}-18\,x^{7\,mn+2\,m^2n-m}$$

The highest degree is 2, so we will use the next prime larger than 3, $P = 5$. Performing our initial evaluation we have

$$a(\omega^2,\omega;x) = 20x^{5\omega^4-2\omega^4+\omega^6+1} + 12x^{7\omega^4-2\omega^4-\omega} - 30x^{7\omega^3+\omega^6+1} - 18x^{7\omega^3+2\omega^4-\omega}$$

As these are the 5^{th} roots of unity, ω^6 is equivalent to ω. Simplifying we have

$$a(\omega^2,\omega;x) = 20x^{3\omega^4+\omega+1} + 12x^{5\omega^4-\omega} - 30x^{7\omega^3+\omega+1} - 18x^{2\omega^4+7\omega^3-\omega}$$

We will now introduce a change to the binomial basis following the same procedure from Section 4.1.

$$\begin{aligned} a(\omega;x) &= 20x^{72\binom{\omega}{4}+108\binom{\omega}{3}+42\binom{\omega}{2}+4\binom{\omega}{1}+1} + 12x^{120\binom{\omega}{4}+180\binom{\omega}{3}+70\binom{\omega}{2}+4\binom{\omega}{1}} \\ &\quad -30x^{42\binom{\omega}{3}+42\binom{\omega}{2}+8\binom{\omega}{1}+1} - 18x^{48\binom{\omega}{4}+114\binom{\omega}{3}+70\binom{\omega}{2}+8\binom{\omega}{1}} \end{aligned}$$

Using the substitution:

$$\begin{aligned} A &= x^{\binom{\omega}{4}} \\ B &= x^{\binom{\omega}{3}} \\ C &= x^{\binom{\omega}{2}} \\ D &= x^{\binom{\omega}{1}} = x^{\omega} \\ E &= x^{\binom{\omega}{0}} = x \end{aligned}$$

we obtain

$$a_t = 20A^{72}B^{108}C^{42}D^4E + 12A^{120}B^{180}C^{70}D^4 - 30B^{42}C^{42}D^8E - 18A^{48}B^{114}C^{70}D^8$$

The polynomial a_t can be factorized into the following product (dropping the monomial factor):

$$a_t = (2A^{72}B^{66} - 3D^4)(5E + 3A^{48}B^{72}C^{28})$$

We now perform back substitution, returning to symbolic polynomials with exponents in the binomial basis.

$$a(\omega; x) = (2x^{72\binom{\omega}{4}+66\binom{\omega}{3}} - 3x^{\omega})(5x + 3x^{48\binom{\omega}{4}+72\binom{\omega}{3}+28\binom{\omega}{2}})$$

For the final step, using the same approach as the extension routine we change to the monomial basis.

$$a(\omega; x) = (2x^{3\omega^4-7\omega^3+4\omega} - 3x^{\omega})(5x + 3x^{2\omega^4-2\omega})$$

Following the steps of Example 6.2.1, an additional 8 evaluation images are re-

quired to calculate the interpolating polynomials for each of the exponents. The GCD and factorization algorithms are detailed in Figures 6.7 and 6.8 respectively. These algorithms are a poor choice for univariate exponents. Obtaining the first image by evaluating at ω will cost exactly the same as running the original change of basis algorithm to completion, but if we interpolate, d more images still need to be calculated.

In practice, we need to change the exponent polynomials to the binomial basis because of fixed divisors; however, there are interesting analytical occurrences if we choose to remain in the monomial basis. It is possible to generate all images from a single point using permutations of the primitive roots of unity. All subsequent images are structurally identical to its generator – the problem of identifying corresponding terms with identical coefficients disappears. Moreover, unlike the change to the binomial basis, sparse exponent polynomials are not mapped to dense polynomials. Performing a single GCD computation rather than $(d+1)^p$ or pdt allows for tremendous performance gains. Additionally, the degree is bound only by the input exponent polynomial coefficient size. While this idea does not work correctly for all possible inputs, the idea that additional images can be generated through permutations of the original is very powerful and something that should be studied further.

sympoly_gcd_unity:
Input: $a, b \in R[n_1, \ldots, n_p; X]$
Output: $g \in R[n_1, \ldots, n_p; X]$ such that g is the GCD of a and b

1. Let d be the max degree of exponent variables in a and b
2. Let P be the next prime larger than $d + 1$
3. Let $\gamma : x_i^{\omega_P^j} \mapsto x_{i,j}$
4. For i from 1 to $(d + 1)^p$ do
   (a) Let $e_i \in \mathbb{C}^p$ be distinct evaluation points for $n_1, \ldots, n_p$, each at powers of ω_P
   (b) $a' \leftarrow a(e_i; X) \quad b' \leftarrow b(e_i; X)$
   (c) Let a' and b' be the result of changing the exponent polynomials of a' and b' to the binomial basis
   (d) $a' \leftarrow \gamma(a') \quad b' \leftarrow \gamma(b')$
   (e) $G \leftarrow \gcd(a', b')$
   (f) $g_i \leftarrow \gamma^{-1}(G)$
   (g) Let g_i be the result of changing the exponent polynomials of g_i to the power basis
5. Return the interpolating symbolic polynomial of $(e_1, g_1), (e_2, g_2), \ldots, (e_{(d+1)^v}, g_{(d+1)^v})$ over X

Figure 6.7: Projection Algorithm for the GCD of Symbolic Polynomials Evaluating at Roots of Unity

sympoly_factor_unity:
Input: $a \in R[n_1, \ldots, n_p; X]$
Output: $f_1, \ldots, f_m \in R[n_1, \ldots, n_p; X]$ such that all f_i are irreducible and their product is a

1. Let d be the max degree of exponent variables in a
2. Let P be the next prime larger than $d + 1$
3. Let $\gamma : x_i^{\omega_P^j} \mapsto x_{i,j}$
4. For i from 1 to $(d + 1)^p$ do
   (a) Let $e_i \in \mathbb{C}^p$ be distinct evaluation points for $n_1, \ldots, n_p$, each at powers of ω_P
   (b) $a' \leftarrow a(e_i; X)$
   (c) Let a' be the result of changing the exponent polynomials of a' to the binomial basis
   (d) $a' \leftarrow \gamma(a')$
   (e) $F_1, \ldots, F_m \leftarrow \text{factor}(a')$
   (f) $f_{i,j} \leftarrow \gamma^{-1}(F_j)$ for $1 \leq j \leq m$
   (g) For $1 \leq j \leq m$, let $f_{i,j}$ be the result of changing the exponent polynomials of $f_{i,j}$ to the power basis
5. Return the interpolating symbolic polynomials of each matching factor $f_{i,j}$ of $(e_1, f_{1,j}), (e_2, f_{2,j}), \ldots, (e_{(d+1)^p}, f_{(d+1)^p,j})$ over X

Figure 6.8: Projection Algorithm for the Factorization of Symbolic Polynomials Evaluating at Roots of Unity

Chapter 7

Empirical Comparison

This chapter will compare the performance of GCDs for four methods (change of basis, sparse interpolation, point selection with dense interpolation, evaluation at roots of unity) empirically.

7.1 Criteria

Each exponent of a symbolic polynomial is a polynomial. Thus, in performing GCDs, there are a multitude of input parameters that must be varied before we can draw any conclusive results.

Increasing the number of base variables has a few impacts on complexity. A multivariate base will more than likely have more exponent polynomials than a univariate. For example, consider the two monomials x^{m^2-1} and $x^{m^2-1}y^{mn^2}$. For point selection, increasing the number of exponent polynomials adds more constraints to the minimization and possibly additional iterations. There is an interpolation for each exponent polynomial in the evaluated images. Increasing the number of exponents will increase the number of interpolations. For change of basis and roots of unity pro-

jection, an additional base variable will linearly increase the number of new variables introduced. Moreover, the GCD computation of the translated Laurent polynomials will have an added complexity with an additional base variable. For our testing, we allow for one, two and three variables in our base.

The number of terms in the base also determines the number of exponent polynomials. With an increased base size sparse GCD methods also become less likely. For our testing, we allow for a small (2 to 4) and large (5 to 9) number of base terms.

The base coefficient ring has no direct impact on the mechanics of the aforementioned algorithms that we are testing, only in the Laurent GCD computations. An increased base coefficient size will have a negative impact on performance; however, as it has no immediate influence over the above procedures, these will be left unvaried. The coefficients are integers ranging from -125 to 125.

The magnitude of the exponent integer-valued coefficients has a much stronger effect on performance. An increased coefficient range will lead to an increased degree when the exponents are evaluated. For our testing, we allow for small (-5 to 5), medium (-10 to 10) and large (-15 to 15) coefficients in the exponent.

Increasing the number of exponent variables has the greatest cost on performance. For the interpolation algorithms, going from a single exponent to several increases the number of required images. It also increases the degrees, when evaluated. Regarding point selection, in the worst case, each additional variable requires a minimization iteration. For our testing, we allow for one, two and three exponent variables.

Along with the number of exponent variables, the degree is also a very important parameter. For projection algorithms, increasing the degree requires more evaluation images. For the extension procedure, more variables are introduced. This is on top of the larger degrees that will result from their evaluation. Larger degrees in the primitive roots of unity projection method leads to larger primes being used. As a

consequence, the transformed degree and number of variables increase drastically. For our testing, we vary the degree from two to five. For this chapter when we say degree we mean the total degree, the highest sum of indeterminate powers in any term.

These parameters were used to randomly generate input for two cases, a trivial and nontrivial GCD. For each case three examples were created. Running these tests against our four routines, gives us over 5000 samples.

All routines were implemented in Maple 11, run serially on a 2 GHz Intel Core Duo with 2 GB ram. Maple 11 uses Brown's modular GCD algorithm for the GCDs of the larger dense projection algorithms; and LinZip, a variation of Zippel's sparse interpolation for the others [Wit04].

7.2 Experimental Results

We will now show several graphs which will summarize our findings from our large sample set. We are not only interested in the timings of these computations, but also in the number of evaluations for interpolation procedures, the number of variables introduced for change of basis, and the maximum degree. Please note that several of the following graphs use a logarithmic Y axis. There are a lot of very small and very large points on the same graph making this suitable.

The implemented point selection procedure does not precisely follow the steps of the algorithm from Figure 6.6. The coded routine uses the difference of the two topmost polynomials (S_1 and S_2 respectively) rather than the topmost and bottommost polynomials (S_1 and S_r respectively) as the objective function. We expect the timing results to be unchanged by this modification.

Each test was given a maximum time allotment of 600 seconds, after which the process was aborted. In the graphs, these occurrences are assigned a value of 1000

seconds. As there are three examples for each distinct parameter combination, the median is always used. This is preferable over the mean because for the point selection and sparse interpolation timings there is sometimes a huge variance between the examples.

The upcoming graphs are for nontrivial GCDs. We always vary the degree along with another parameter (for each degree, rather than stacked atop each other, the parameters are spaced horizontally to facilitate comparison).

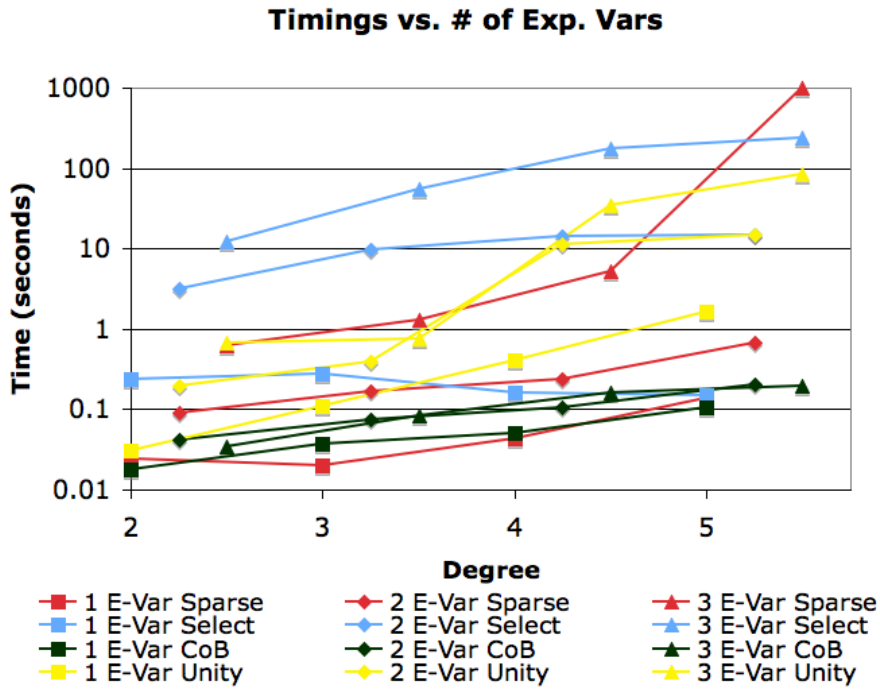
Varying the Number of Exponent Variables

We alter the number of exponent variables and degree, while keeping the base size small, exponent coefficient size small and a univariate base.

For univariate exponents, all algorithms complete quickly. From Chapter 6, we know that the roots of unity projection algorithm is not optimal for single variable exponent polynomials. For the given parameter values, the number of exponent variables seems to have no dramatic effect on the change of basis results. The overhead of the point selection procedure, especially for multivariate exponents for small examples contributes to its early poor performance; however, notice that the point selection method is not growing as fast as the sparse interpolation technique and is better for the largest example in this set.

The next graph shows the problem in selecting evaluation points at random. Sparse interpolation was reaching degrees close to a million. Roots of unity projection was also generating large image degrees. Point selection can bring the degree down to the thousands.

Notice the step increases in timings, image degree and number of variables between degrees of 3 and 4 for the primitive roots of unity projection procedure. The examples



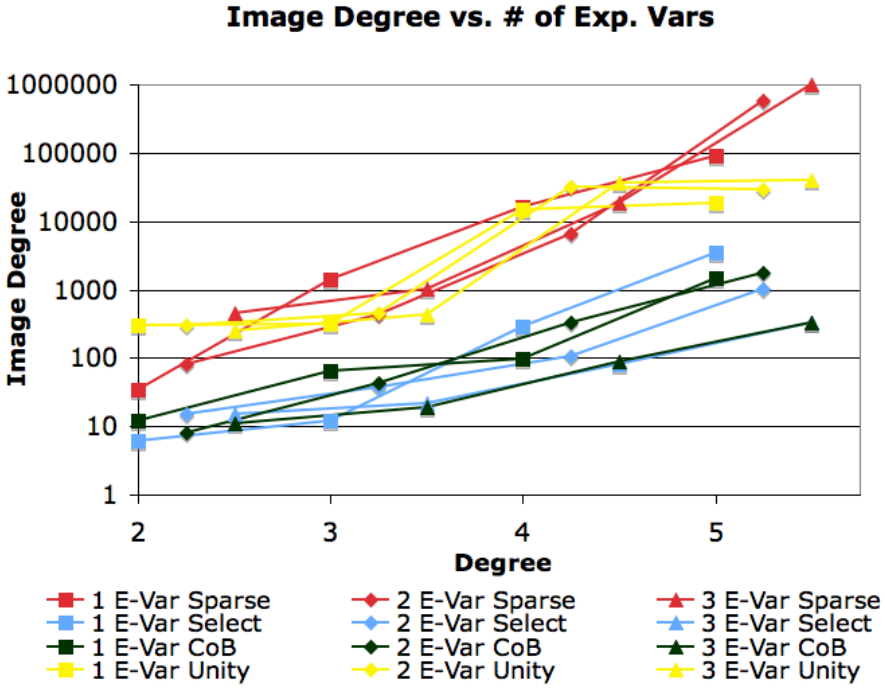
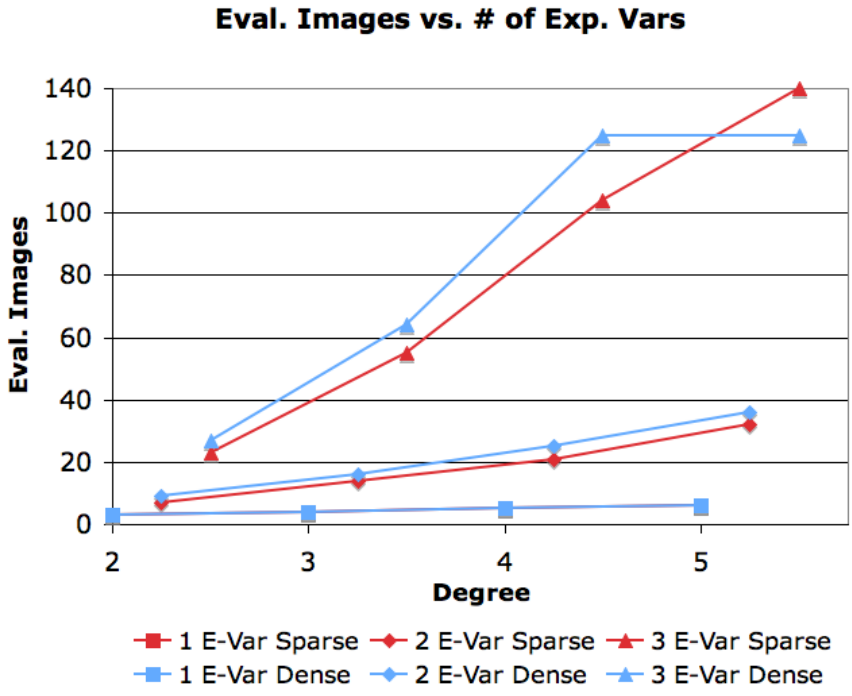


Figure 7.1: Experimental Results by Varying Number of Exponent Variables and Degree



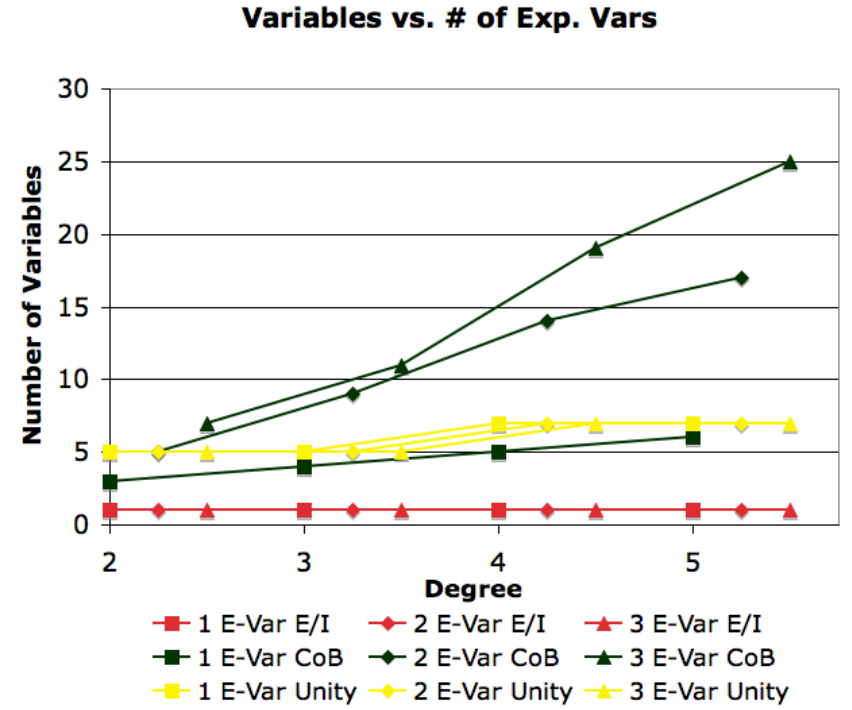


Figure 7.2: Experimental Results by Varying Number of Exponent Variables and Degree

for degree values of 2 and 3 will use the prime 5 – as they both use the same prime the results are nearly identical. Alternatively, when the degree is 4 or 5, the prime 7 is used. The increased prime leads to larger degrees and an increased number of variables. The step increases, which can be seen in other computer algebra techniques evaluating at roots of unity, continue for larger degrees. A step to the next prime occurs when the degree is one less than the current prime (ie. 2, 4, 6, 10, 12, 16, and so on).

In looking at the extreme cases of this sample set we note that the extension algorithm can introduce upwards of 25 total variables in its images, while interpolation requires more than 100 images. It appears that performing one GCD with 25 variables is much more efficient than 100 univariate GCDs. Even discarding the time it takes to find the optimal evaluation images, the sparse and dense projection methods can not come close to matching the efficiency of the change of basis routines.

Varying the Number of Base Variables

We alter the number of base variables and degree, while keeping the base size small, exponent coefficient size small and a bivariate exponent. The results are shown in the graphs from Figure 7.3 and Figure 7.4.

One can draw many of the same conclusions from Figures 7.1 – 7.2 and Figures 7.3 – 7.4. We see that change of basis is still the fastest. The remaining three are a few orders of magnitude worse. The degrees of the sparse evaluations are much larger, while the optimal evaluations remain at approximately one thousand. This attributes to the very poor performance of using random evaluations for several base variables.

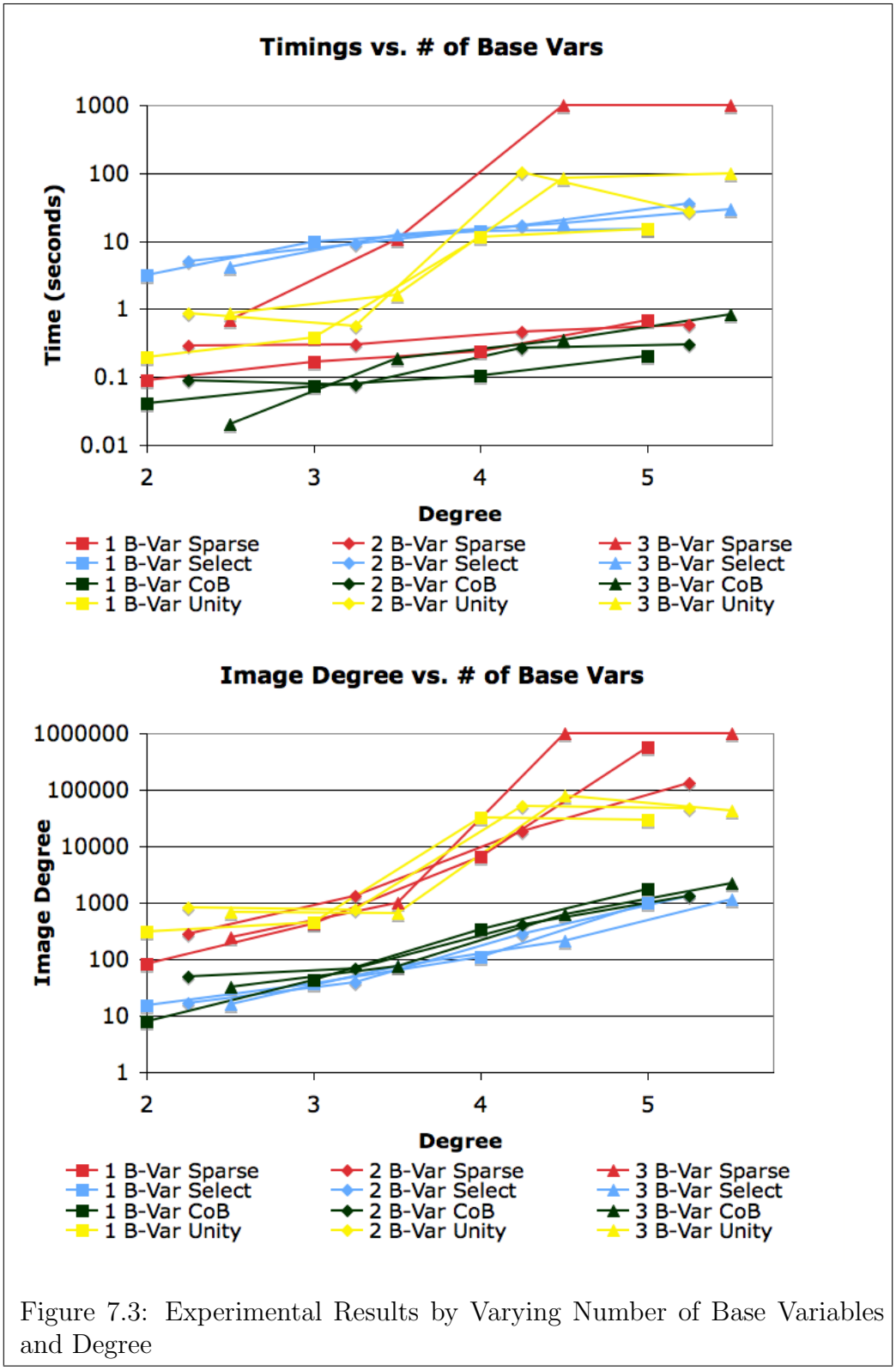
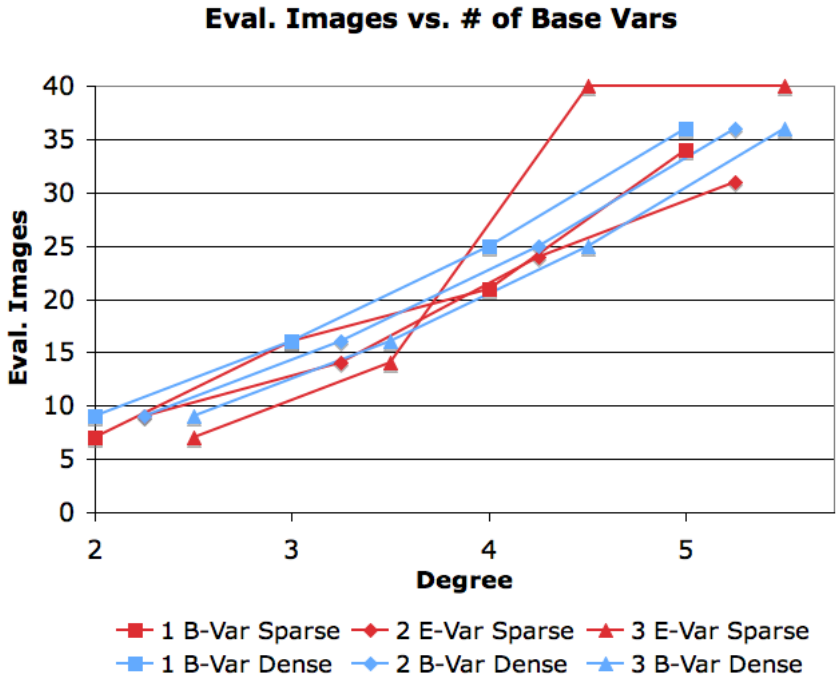


Figure 7.3: Experimental Results by Varying Number of Base Variables and Degree



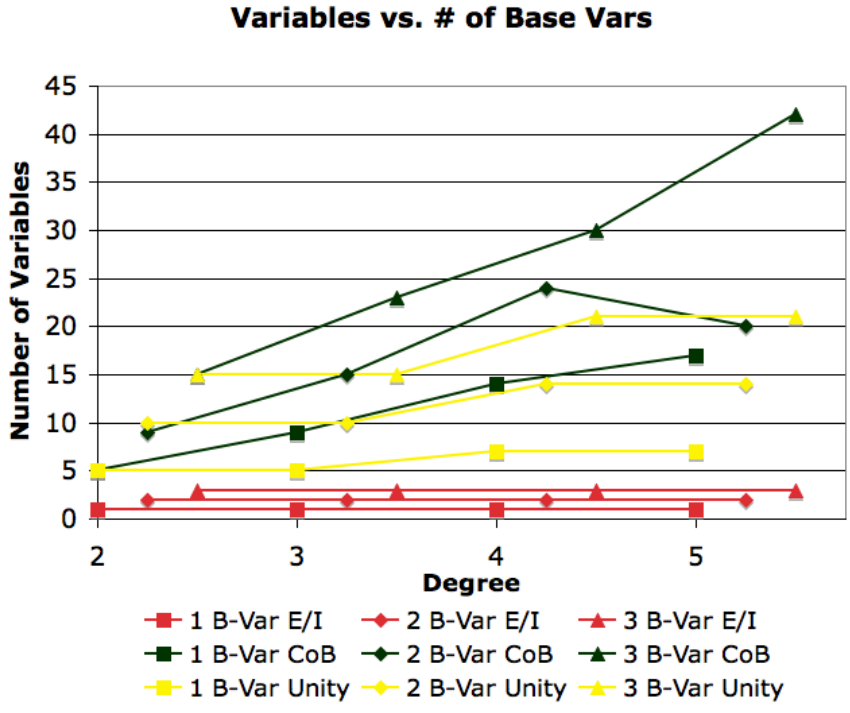


Figure 7.4: Experimental Results by Varying Number of Base Variables and Degree

Varying the Size of the Exponent Coefficients

We alter the size of the exponent coefficients and degree, while keeping the base size small, univariate base and bivariate exponent. The results are shown in the graphs from Figure 7.5.

The size of the exponent coefficients has no effect on the number of evaluation points or the number of variables required; their graphs have been omitted. Aside from the sparse medium points, the data sets for each algorithm are nearly indistinguishable. For the given parameter values, modifying the exponent coefficients did not have a noticeable performance change.

Varying the Number of Base Terms

We alter the number of terms in the base and degree, while keeping a univariate base and bivariate exponent with small coefficients. The results are shown in the graphs from Figure 7.6

One would expect the degree of images with additional base terms to remain unchanged; however, a small increase is seen. The number of base terms is proportional to the number of exponent polynomials. When evaluating at random, input with more exponent polynomials are more likely to contain bad evaluations, which lead to high degrees. The point selection optimization routine would have additional constraints, making it more difficult to find a minimization.

The timings are similar to previous tests with the change of basis procedure besting the others by a couple orders of magnitude. The point selection algorithm is more efficient than sparse interpolation for degrees larger than three and roots of unity projection for degrees larger than four.

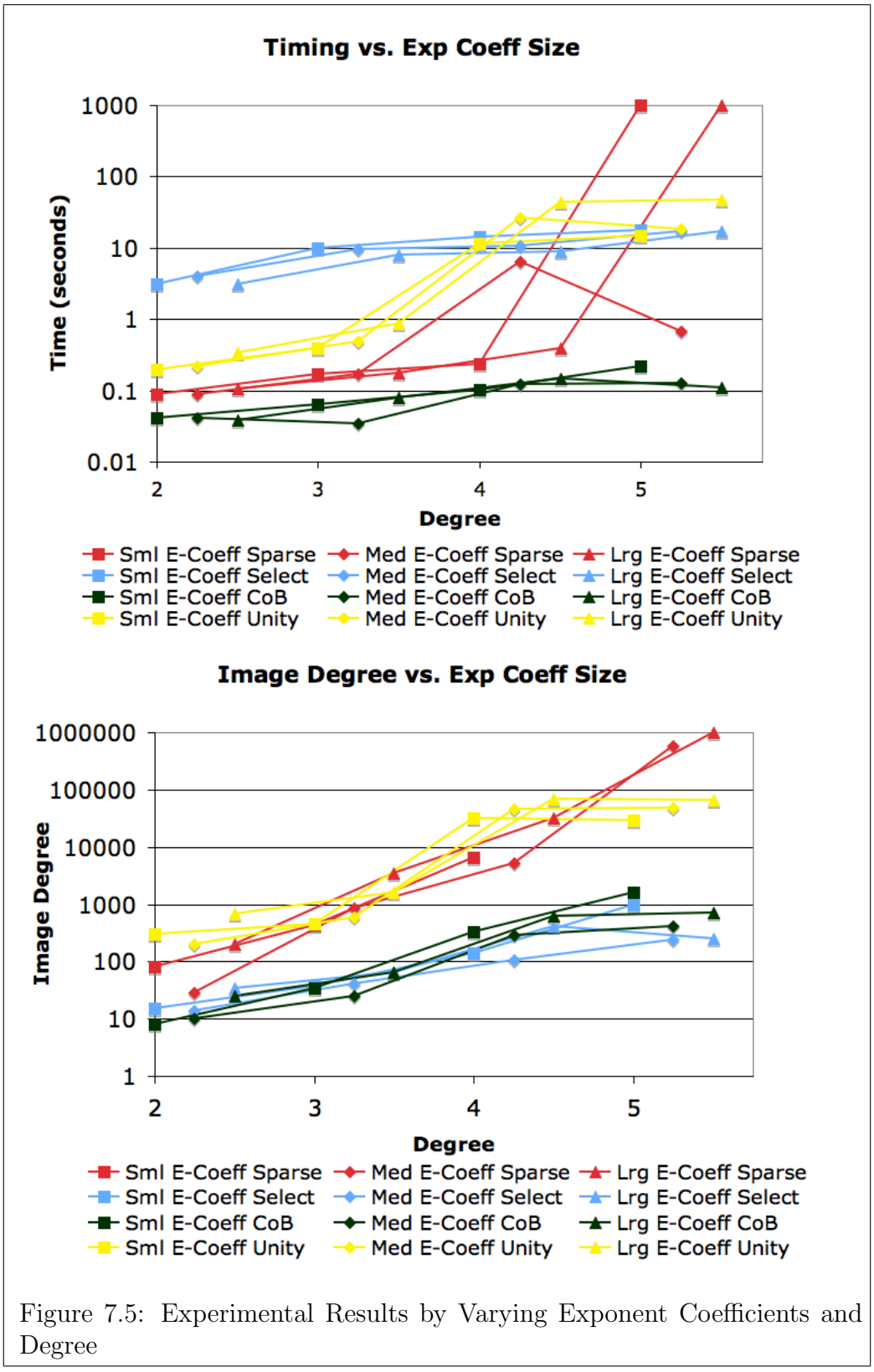


Figure 7.5: Experimental Results by Varying Exponent Coefficients and Degree

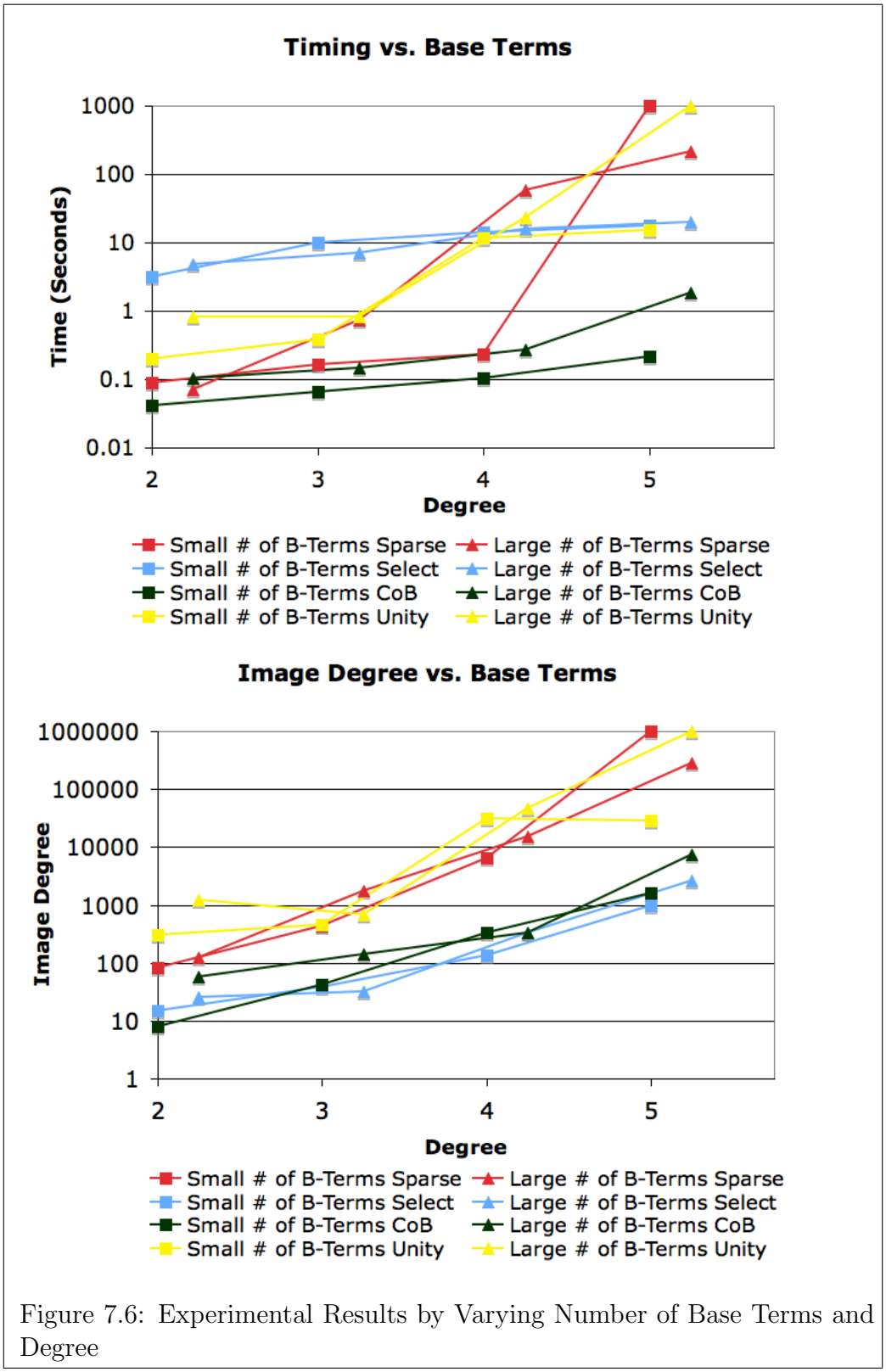


Figure 7.6: Experimental Results by Varying Number of Base Terms and Degree

Larger Examples

The exponent degree and number of variables have the greatest impact on performance. These two parameters will be varied once again, this time with three base variables, a large base and small coefficient size. The results are shown in the graphs of Figure 7.7 and Figure 7.8.

Note the tremendously high rate of increase for sparse interpolation in both timing and image degree for small input degrees. The cost of going to the next prime for the roots of unity projection method is also very large. Point selection is more efficient than the other evaluation/interpolation methods for degrees larger than 2. The extension algorithm performs much better than its competition. It can complete problems in less than 10 seconds that take more than 10 minutes for its projection counterparts.

The largest examples for change of basis introduced close to 100 new variables and had a degree of around 10000, and computed a single GCD. Point selection computes over 100 GCDs with three variables with a degree of around 25000. The GCD algorithm used by Maple is much better suited for large number of variables than large degree.

Trivial GCDs

The findings of the trivial GCD experiments seen in Figures 7.9 – 7.11 are similar to their respective nontrivial complements. By definition, these GCDs are one. As such, we have not measured the degree or number of variables present in the output. The sparse interpolation and roots of unity projection methods show the most noticeable differences. Sparse interpolation only needs to interpolate once, requiring $d+1$ images. This is especially evident for the smaller examples of Figures 7.9 – 7.11, where the

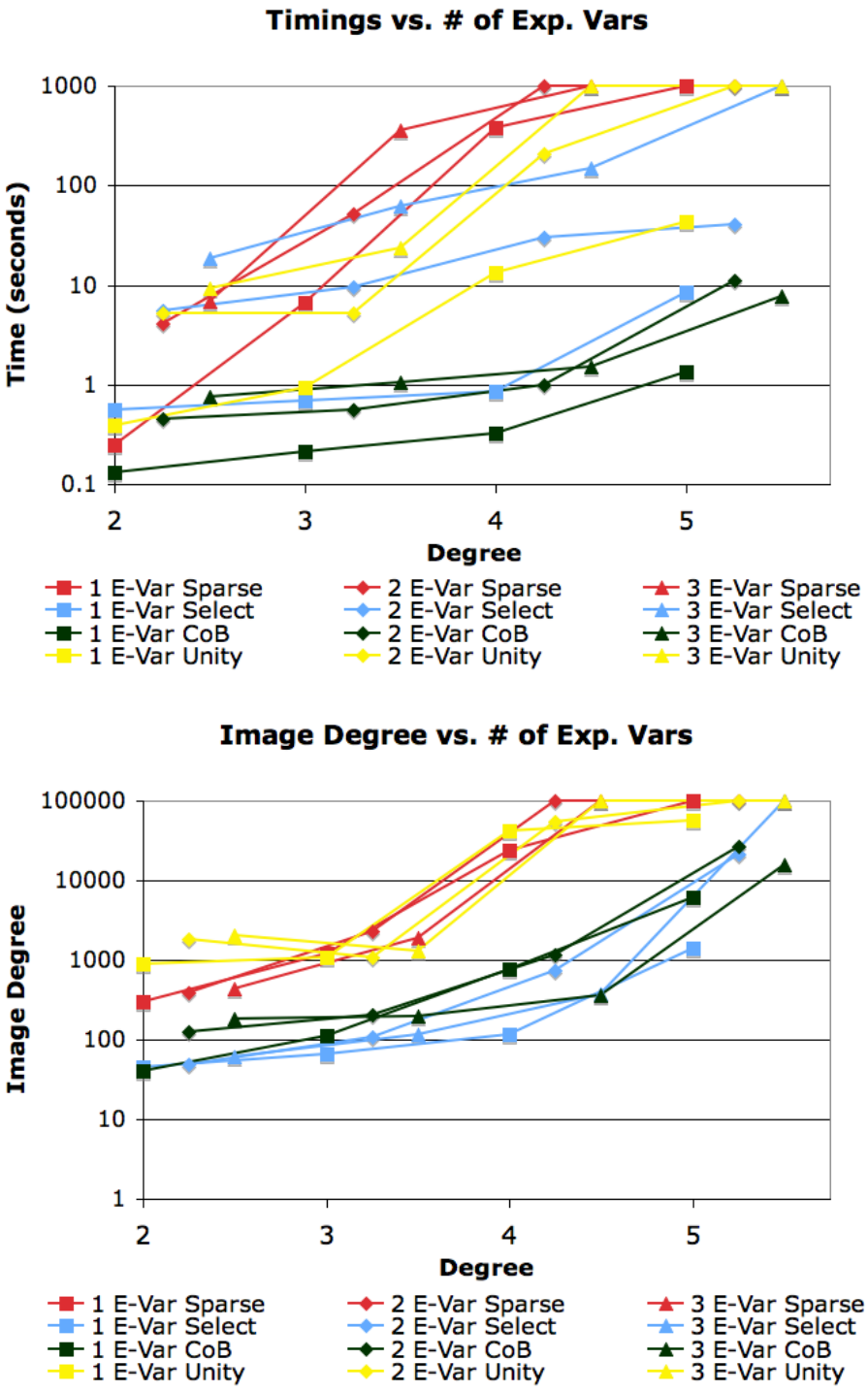
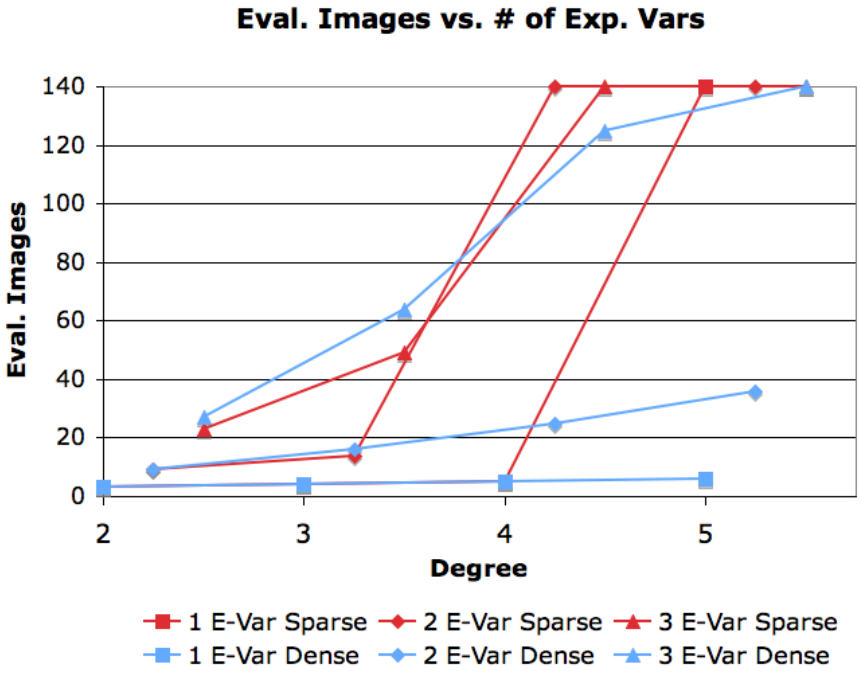


Figure 7.7: More Experimental Results by Varying Number of Exponent Variables and Degree



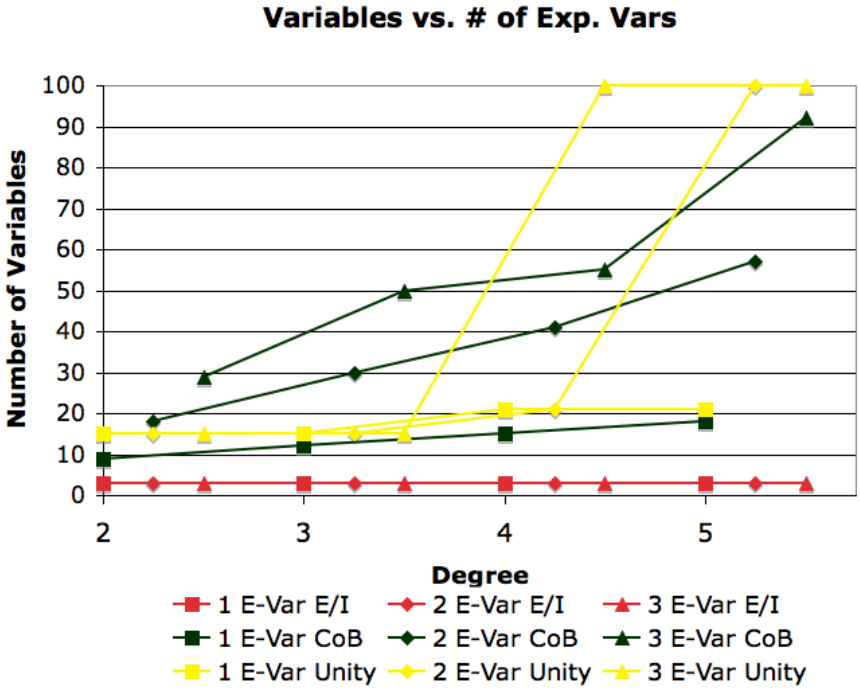


Figure 7.8: More Experimental Results by Varying Number of Exponent Variables and Degree

sparse scheme outperforms extension.

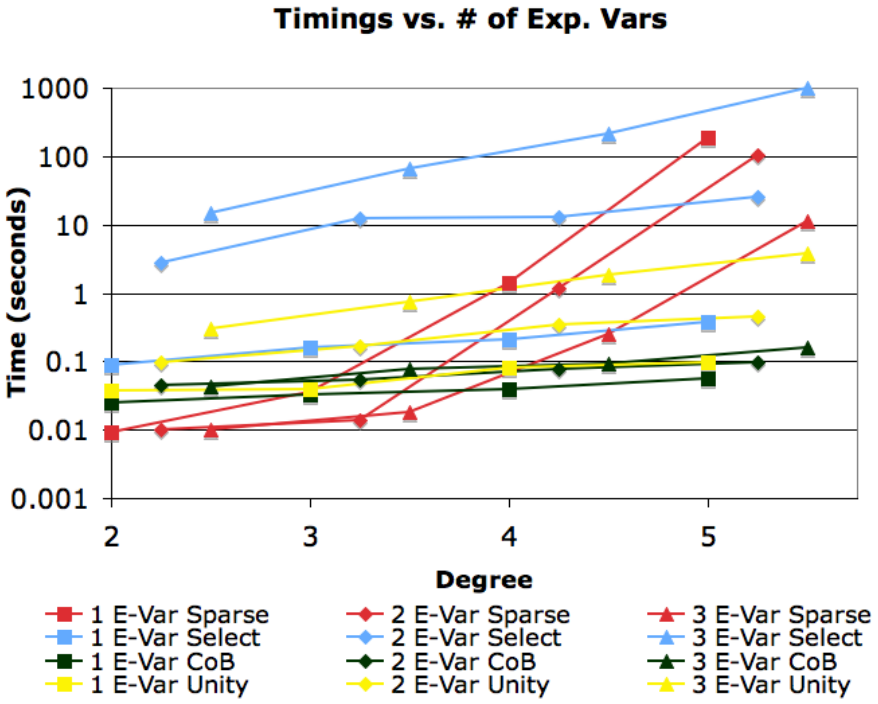
7.3 Analysis of Results

It is clear that for GCDs, change of basis is the most suitable algorithm. In practice, when the varied criteria were small, compared to random selection, the overhead of searching for optimal evaluations outweighed its benefits. As the exponent degree rises, and more variables are present, selecting good evaluations becomes increasingly more important. Comparing the projection methods we see that the point optimization algorithm is the best, followed by roots of unity projection and then sparse interpolation.

In the set of sparse interpolation examples, it was not uncommon for the system to be computing the very first evaluated GCD for over 10 minutes, often consuming all available system memory until the Maple kernel crashed. It seems odd that a GCD calculation would require upwards of several gigabytes of memory. Upon further investigation of a typical kernel crash, the Maple Chinese remaindering of the GCD coefficients was using more than 700 moduli. The cumulative product of these moduli had reached over 3000 digits in length.

The degree of the change of basis algorithm was generally the same order as the point selection method. The number of variables it introduces increases at about at the same rate as the number of images needed for dense interpolation. As aforementioned, the GCD algorithm favors performing one GCD with many variables over many GCDs with few variables.

In most examples, we saw a reduced number of required evaluations for the sparse method. The examples were not very sparse, so the numbers were sometimes close to the naive case. Regardless of sparsity, change of basis was consistently the fastest – it



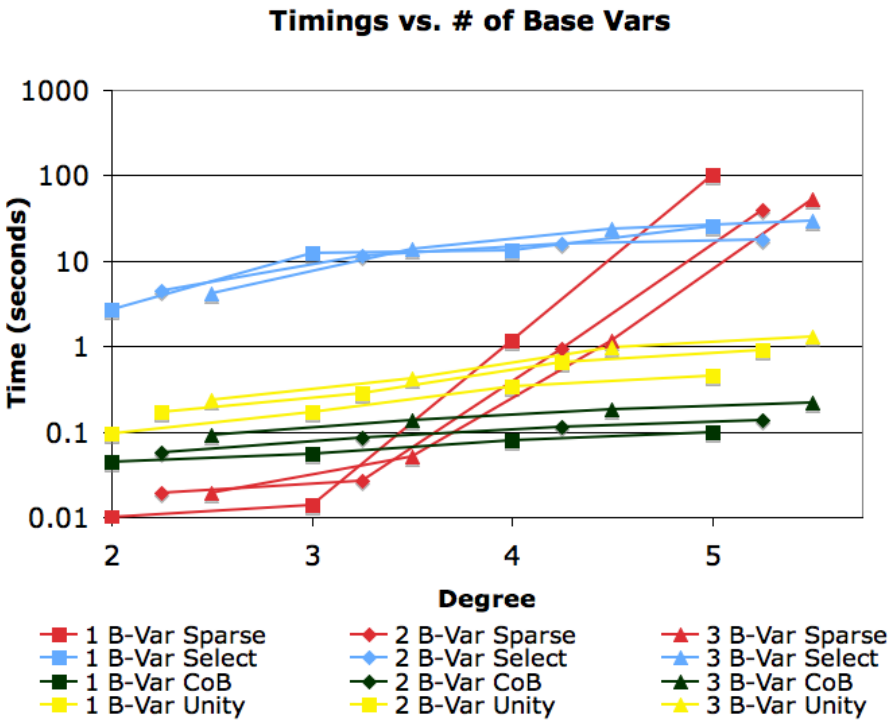
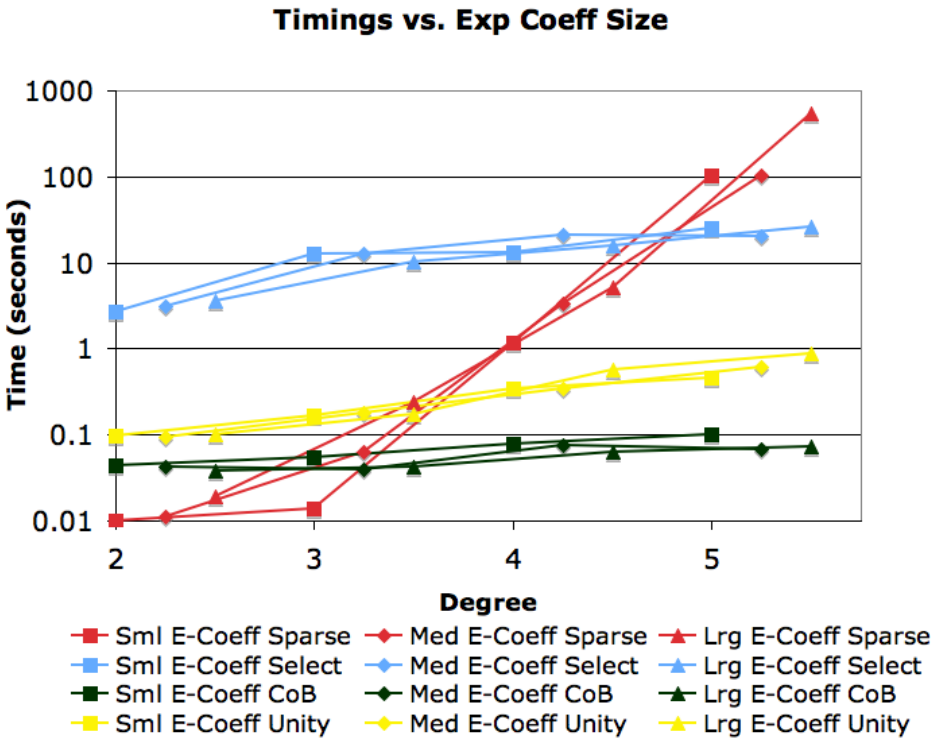


Figure 7.9: Experimental Results for Trivial GCDs



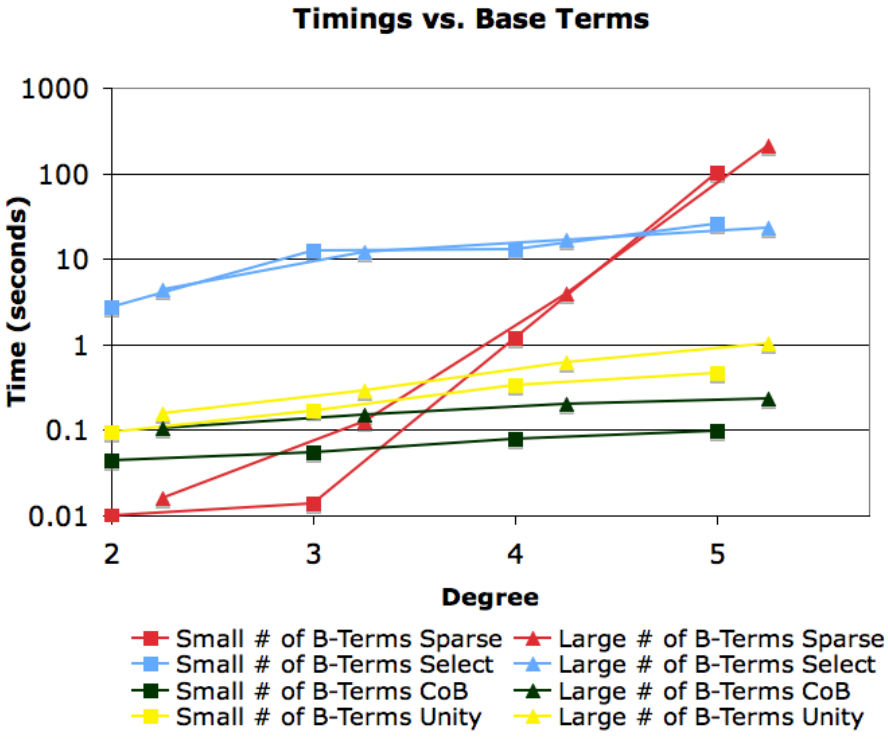


Figure 7.10: Experimental Results for Trivial GCDs

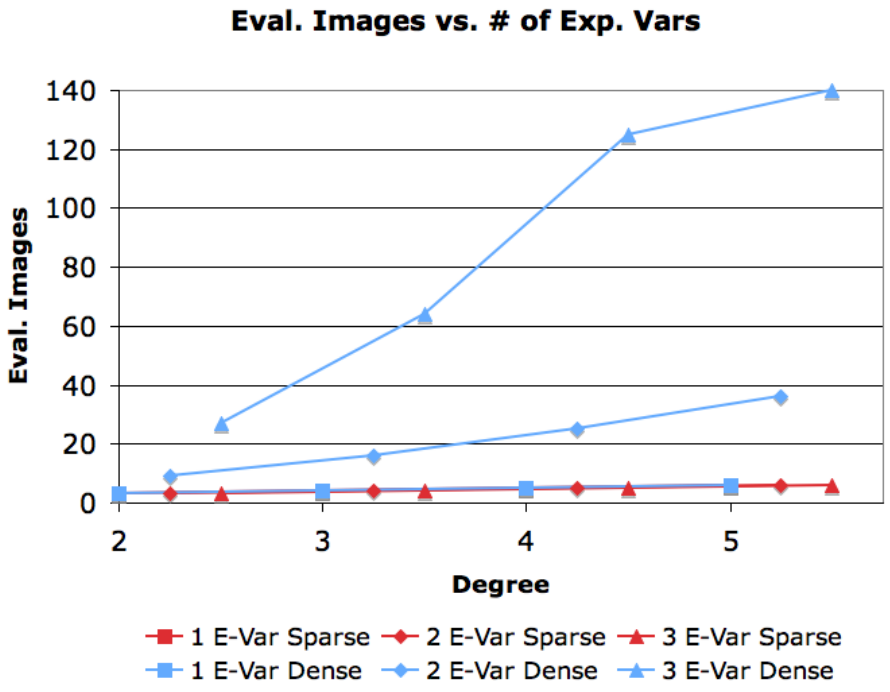


Figure 7.11: Experimental Results for Trivial GCDs

is the most suitable general purpose algorithm to solve GCDs of symbolic polynomials.

The next chapter will complement these findings by presenting a formal theoretical complexity analysis of the algorithms.

Chapter 8

Algorithmic Complexity

This chapter outlines the general asymptotic complexity estimates of the four methods tested in the previous chapter.

We will give complexity estimates for the extension, evaluation/interpolation, point selection, and roots of unity projection algorithms. We will make the assumption that all input, intermediate and output exponent, coefficient and evaluation integers can fit into a machine word.

Experimenting with the sparse interpolation scheme, we have seen the great variance of timings from similar instantiations. Providing a thorough work estimate is not possible. The number of operations varies greatly according to the structure and coefficients of its input. Nonetheless, a general estimate can be given.

As illustrated throughout the previous chapters, symbolic polynomials have a number of varying parameters. They will be denoted as follows:

- p - The number of exponent variables
- v - The number of base variables

- d - Maximum degree of input exponent polynomials
- D - Maximum degree after an extension or evaluation
- c - Maximum exponent coefficient
- C - Base coefficient size (logarithmic)
- T - The number of base terms
- t - The maximum number of terms of any exponent polynomial
- e - The number of exponent polynomials
- n - The number of variables after an extension or evaluation

We will assume that the GCD computation of the extended and evaluated images uses Brown's and Zippel's (LinZip) algorithms for dense and sparse GCDs respectively. The work complexity for dense polynomial GCDs is $O(C^2D^n + nCD^{n+1})$. We can assume that the number of variables, n, and the degree, D, are much larger than the coefficient length. Thus, the estimate simplifies to $O(nD^{n+1})$. The work complexity for sparse polynomial GCDs is $O(C^2T^2+CTD^2+TnD^3+CnDT+D^2n^2T+CT^3+T^3nD)$. Once again we can assume that the number of variables, n, and the degree, D, are significantly greater than the number of terms and the coefficient length. This simplifies the estimate to $O(nD^3 + n^2D^2)$. [Wit04]

8.1 Extension Algorithm

The change of basis method can be divided into three sections: mapping the input to a Laurent polynomial, a GCD calculation and mapping back to a symbolic polynomial.

We will first calculate the number of variables introduced. We assume that p and v are both equal to 1. A univariate exponent polynomial of degree d, when rewritten in the binomial basis will generally have $d+1$ terms, regardless of initial sparsity, and thus have $d+1$ new variables. If we now allow for an arbitrary number, p, of exponent variables, there would be $p(d+1)$ exponent translations, namely $\binom{n_i}{j}$, $i = 1, \ldots, p; j = 0, \ldots, d-1$. The expansion of the binomial polynomial will have a term for each product of the form $\binom{n_1}{d_1}\binom{n_2}{d_2}\cdots\binom{n_p}{d_p}$. There are $(d+1)^p$ such terms. Through algebraic independence, a new variable is created for each of these terms. We will now allow for an arbitrary number of base variables. The $(d+1)^p$ terms will be in the exponents of each base variable. The algebraic independence of x^n and y^n for $x \neq y$ gives $(d+1)^p$ new variables for each base indeterminate, totaling $n \in O(v(d+1)^p)$.

The mapping $\gamma : x_k^{\binom{n_1}{i_1}\cdots\binom{n_p}{i_p}} \mapsto x_{k,i_1\ldots,i_p}$, which is used to change from a symbolic polynomial to a Laurent polynomial outputs n new variables, and γ^{-1} inputs n variables. Therefore the first and last step of the extension algorithm together have a complexity of $O(v(d+1)^p)$.

Note that the degree of the image is the maximum coefficient of all the exponent polynomials in the binomial basis. Recall the equation $\binom{x}{n} = \frac{x(x-1)\ldots(x-n+1)}{n!}$. In mapping to the binomial basis, the factorial in the denominator will shift to the numerator increasing the coefficient by a factor of $O(d!)$. The largest possible coefficient (and image degree) is then $D \in O(cd!)$.

The new exponent polynomials are almost always dense, making the extended images even more sparse. Using LinZip we get a final estimate of

$$Cost_{CoB} = O(n + nD^3 + n^2D^2) = O(v(d+1)^p(cd!)^3 + (v(d+1)^p)^2(cd!)^2)$$

8.2 Sparse Interpolation

The symbolic polynomial sparse interpolation method of Chapter 5 performs $O(pdt)$ polynomial GCDs followed by e sparse polynomial interpolations.

The number of variables of all evaluated images remains unchanged; $n = v$.

Let α be an evaluation of exponent indeterminates. The degree bound of that evaluation will be $O(c\alpha^d)$. In general we try to select α as small as possible. Following this idea, the best values for each indeterminate would be in the interval $[-\frac{d+1}{2}, \frac{d+1}{2}]$. The degree bound is $D \in O(c((d+1)/2)^d)$.

Recall Zippel's sparse interpolation algorithm of Figure 2.2. A total of $O(pd)$ systems of t linear equations in R are solved. A total of $O(pt)$ univariate interpolating polynomials are calculated. Assuming each system of equations is solved by Gaussian elimination in $O(n^3)$, and each Lagrange interpolation is done in $O(n^2)$, the cost of a single sparse interpolation is $O(pdt^3)$.

From the estimate of D, we can safely assume that $(D+1)^n \gg t$. The complexity of the sparse interpolation scheme, $Cost_{Sp}$, is

$$\begin{aligned} Cost_{Sp} &= O(pdt(nD^3 + n^2D^2) + e(pdt^3 + pt\log^2(d)) \\ &= O(pdt(vc^3((d+1)/2)^{3d} + v^2c^2((d+1)/2)^{2d}) + epdt^3) \end{aligned}$$

8.3 Optimal Evaluations

The effectiveness of the point selection routine of Figure 6.6 on reducing image degrees depends entirely upon the set of exponent polynomials, and cannot be generalized by just their degree and coefficient bound. Worst case scenario is that the evaluations are the same as selecting at random, or at small values. It is however, much more

difficult to uncover a non-experimental estimate for the average and best cases.

As the evaluations returned from this procedure are all independent of each other, they can not be used with Zippel's interpolation. A typical GCD calculation using this method would then consist of finding $(d+1)^p$ "good" evaluations followed by their GCDs and e multivariate interpolations.

The number of times the minimization is repeated, m in Figure 6.6, is normally taken to be linearly related to the degree. For every exponent variable, a gradient descent is done. Steepest descent has a linear convergence in the degree, and each iteration costs $O(d^2)$. This results in an end cost of $O(vd^4)$.

As previously stated, we can not give an accurate estimate for D, and like the sparse projection method, the number of variables remains unchanged.

Each multivariate interpolation is done using the extended Vandermonde Matrix idea from Chapter 2. This is equivalent to solving a system with $(d+1)^p$ equations and unknowns. Assuming this is solved by Gaussian elimination, the cost will be $O((d+1)^{3p})$.

For the same reasons as the last section, we can safely assume the evaluated images are sparse. Using LinZip for the image GCD calculations, we get a final asymptotic estimate, $Cost_{PSel}$, of

$$\begin{aligned} Cost_{PSel} &= O(vd^4 + (d+1)^p(nD^3 + n^2D^2) + e(d+1)^{3p}) \\ &= O(vd^4 + (d+1)^p(vD^3 + v^2D^2) + e(d+1)^{3p}) \end{aligned}$$

8.4 Roots of Unity Projection

The roots of unity projection algorithm of Figure 6.7 performs $(d+1)^p$ change of basis GCDs followed by e multivariate interpolations in $\mathbb{C}$.

Let P be the next prime larger than $d+1$. We will assume that $P \in O(d)$. The evaluated exponent polynomials, regardless of p, become univariate after evaluating at primitive roots of unity. Each exponent polynomial will have at most P terms. The maximum number of terms will remain unchanged by changing to the binomial basis. We are using the extension method from Section 8.1 to change basis, therefore $n \in O(v(d_e+1)^{p_e})$, where p_e and d_e are the number of exponent variables and degree after evaluating at roots of unity. We have $d_e \in O(P)$ and $p_e = 1$, giving us $n \in O(vd)$ indeterminates.

Similarly, we can use the same reasoning from Section 8.1 to calculate D from d_e. The evaluated exponent polynomials have a degree, d_e, in $O(d)$. Therefore, as $D \in O(cd_e!)$, we have $D \in O(cd!)$. Although this is the same complexity class as the degree for the change of basis method, $O(cd!)$ is its lower bound for when each exponent variable is evaluated at the first power of ω_P. The degree of the roots of unity projection method are always larger than the change of basis.

The assumption $P \in O(d)$ is a general estimate used in lieu of just P to facilitate comparisons with the other methods. If we ignore this assumption, we have $n \in O(vP)$ and $D \in O(cP!)$. We see that the number of variables and translated degree only change when a larger P is needed. This attributes to the step increases of the roots of unity projection method discovered in the previous chapter.

The evaluation points can be organized similar to that of Zippel's interpolation. We can exploit this to use a better recursive dense multivariate interpolation algorithm. This calculates $O(d+1)^p$ univariate interpolations in $\mathbb{C}$ of size $O(d)$ totaling $O((d+1)^p \log^2(d))$. This is repeated e times, once for each exponent polynomial in the GCD images.

Asymptotically, and usually in practice, the images are sparse. The complexity of

roots of unity projection with the LinZip algorithm, $Cost_{Un}$, is

$$\begin{aligned} Cost_{Un} &= O((d+1)^p(n + nD^3 + n^2D^2) + e(d+1)^p \log^2(d)) \\ &= O((d+1)^p(vd(cd!)^3 + (vd)^2(cd!)^2 + e\log^2(d))) \end{aligned}$$

8.5 Summary of Results

The findings of the previous sections are summarized in Table 8.1.

Method	Asymptotic Complexity
CoB	$O(v(d+1)^p(cd!)^3 + (v(d+1)^p)^2(cd!)^2)$
Sp	$O(pdt(vc^3((d+1)/2)^{3d} + v^2c^2((d+1)/2)^{2d}) + e(pdt^3 + pt\log^2(d)))$
PSel	$O(vd^4 + (d+1)^p(vD^3 + v^2D^2) + e(d+1)^{3p})$
Un	$O((d+1)^p(vd(cd!)^3 + (vd)^2(cd!)^2 + e\log^2(d)))$

Table 8.1: Asymptotic Complexity Estimates for Symbolic Polynomial GCDs

Each algorithm noted above is exponential with respect to the number of exponent variables. The $(d+1)^p$ multiplier of the roots of unity projection method would be replaced by a linear factor if sparse interpolation were employed. Similarly, assuming that D is non-exponential, it would be theoretically possible to reduce the point selection complexity to polynomial time if the optimized evaluation values could allow for Zippel interpolation. However, this is probably not a safe assumption; according to the experimental evidence, its degree grows exponentially, although several times more slowly when values are selected at random.

Let us compare the unity projection algorithm against the change of basis method for when $p = 1$. Simplifying the two complexities, we see

$$\begin{aligned} Cost_{CoB} &= O(vd(cd!)^3 + (vd)^2(cd!)^2) \\ Cost_{Un} &= O(d(vd(cd!)^3 + (vd)^2(cd!)^2) + de\log^2(d)) \end{aligned}$$

For univariate exponents, the reduced number of variables for the roots of unity projection method disappears. The cost of computing each image is the same as the standard change of basis procedure. As $O(d)$ images are required to interpolate, $Cost_{Un}$ is d times larger than $Cost_{CoB}$.

In practice, these theoretical complexities may not always agree with empirical data. It is sometimes the case in Maple that the sparse interpolation and point selection GCD images are calculated using Brown's algorithm over LinZip. This leads to doubly exponential complexity and explains the appreciable timing difference against the change of basis method. Brown's modular algorithm is normally invoked only in the larger inputs of the example suite. It is unclear why this occurs; nonetheless, hard coding Maple to explicitly perform LinZip in these instances does not produce preferable results.

Chapter 9

Conclusion

9.1 Summary

Within this discourse, we have analyzed two existing methods for solving operations on symbolic polynomials, and have investigated three new, alternative routines. In doing so, we hope to have shed some light on the symbolic computational inadequacies in modern computer algebra software.

The change of basis extension routine naturally arose from the UFD and algebraic independence proofs of symbolic polynomials. On account of the binomial basis, exponent polynomials are inherently dense leading to an exponential number of base variables. The degree bound is also exponential.

Evaluation and interpolation is the second approach to handling symbolic polynomials. Watt [Wat06] has given a dense interpolation method which has a constant number of variables, but uses an exponential number of evaluation images of non-polynomial degree. The number of evaluation images is reduced to a linear scale by implementing a sparse interpolation scheme. This improves runtime performance, but the high degree still leads to intractability for moderate to large examples.

We can decrease the degree of the evaluated images by selecting the values that will minimize the maximal difference of all exponent polynomials. For smaller examples, the cost of searching for these values outweighs the benefits, but for more complex input, this method of evaluation optimization is advantageous.

Another way to bound the degree of the images is to evaluate at primitive roots of unity and perform a similar substitution of variables as the change of basis method. When evaluating at different powers of ω, we can reduce multivariate exponent polynomials to univariate. The binomial expansion of the extension method introduces a linear, rather than exponential, number of new variables. Using the extension method leads to an unbounded degree that grows exponentially.

The above procedures were tested with a number of semi-sparse examples. For moderate sized input symbolic polynomials we see sparse interpolation perform the worst, followed closely by the optimized value selection routine and primitive roots of unity projection. The change of basis method is orders of magnitude more efficient. All of the evaluation/interpolation methods have fewer variables, but the degrees are larger. Additionally, a multitude of image GCDs and interpolations need to be calculated. Although it does not retain sparsity, in practice the change of basis is the most suitable algorithm from both a design and runtime perspective.

9.2 Future Work

There are a number of questions that remain unanswered and ideas to be explored.

The evaluation/interpolation algorithms all exhibit many independent processes: calculating several GCDs and interpolating each multivariate exponent polynomial. A multithreaded parallel implementation of symbolic polynomial interpolation would see good speed up for roots of unity projection. This is on top of the added benefit

of a parallel GCD algorithm for the images.

There are a couple of alternative implementations that would most likely see some performance gains. The optimal point selection method could return values ordered in a way to allow for Zippel interpolation. This would lower the number of images to a linear scale, however, this addition would still not be comparable to the extension method. On the other hand, the roots of unity projection values are already ordered in a way to allow for sparse interpolation. This would eliminate the exponential factor in the complexity estimate.

It may be possible for symbolic polynomial projection algorithms to be more efficient than extension algorithms if additional images can be generated through permutations of an original. Mentioned in the closing remarks of Chapter 6, this may be a worthwhile investigation.

Recall that the primary mathematical objects studied in this thesis, described by the ring $R[X;Y]$, are just one specific instance of symbolic polynomials. The exponents need not be integer-valued polynomials. The base need not be an indeterminate. Symbolic exponents could appear on coefficients. It would be worthwhile to look at these issues.

Bibliography

[CCS98] Chabert, Jean-Luc, Chapman, Scott T., and Smith, William W. The Skolem property in rings of integer-valued polynomials. *Proceedings of the American Mathematical Society*, 126(11):3151–3159, nov 1998.

[Cha93] Jean-Luc Chabert. Integer-valued polynomials, Prufer domains, and localization. *Proceedings of the American Mathematical Society*, 118(4):1061–1073, aug 1993.

[CM99] Jianer Chen and Antonio Miranda. A polynomial time approximation scheme for general multiprocessor job scheduling (extended abstract). In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 418–427, 1999.

[DFJ54] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[Fau03] Laurene Fausett. *Numerical Methods: Algorithms and Applications*. Pearson Education Inc., 2003.

[Fri96] Sophie Frisch. Integer-valued polynomials on Krull rings. *Proceedings of the American Mathematical Society*, 124(12):3595–3604, dec 1996.

[GHLS90] Gilmer, Robert, Heinzer, William, Lantz, David, and Smith, William. The ring of integer-valued polynomials of a Dedekind domain. *Proceedings of the American Mathematical Society*, 108(3):673–681, mar 1990.

[HRS89] C.W. Henson, L. Rubel, and M. Singer. Algebraic properties of the ring of general exponential polynomials. *Complex Variables Theory and Applications*, 13:1–20, 1989.

[MB79] Saunders MacLane and Garret Birkhoff. *Algebra*. Macmillan Publishing Co., Inc., 2nd edition, 1979.

[PW06] Wei Pan and Dongming Wang. Uniform Gröbner bases for ideals generated by polynomials with parametric exponents. In *ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 269–276, New York, NY, USA, 2006. ACM.

[Tur86] Jan Turk. The fixed divisor of a polynomial. *The American Mathematical Monthly*, 93(4):282–286, apr 1986.

[vzGG03] Joachim von zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.

[Wat06] Stephen Watt. Making computer algebra more symbolic. In *Transgressive Computing*, pages 43–49, 2006.

[Wat07a] Stephen Watt. Two families of algorithms for symbolic polynomials. In I. Kotsireas and E. Zima, editors, *Computer Algebra 2006: Latest Advances in Symbolic Algorithms – Proceedings of the Waterloo Workshop*, pages 193–210. World Scientific, 2007.

[Wat07b] Stephen Watt. What happened to languages for symbolic mathematical computation. In *Programming Languages for Mechanized Mathematics (PLMMS)*, 2007.

[Wit04] Allan Wittkopf. *Algorithms and Implementation for Differential Elimination*. PhD thesis, Simon Fraser University, 2004.

[Yok04] Kazuhiro Yokoyama. On systems of algebraic equations with parametric exponents. In *ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 312–319, New York, NY, USA, 2004. ACM Press.

[Zip79] Richard Zippel. *Probabilistic Algorithms for Sparse Polynomials*. PhD thesis, Massachusetts Institute of Technology, 1979.

VITA

Name: Matthew Malenfant

Born: New Glasgow, Nova Scotia, 1983

Eduation:

- St. Francis Xavier University

  Antigonish, Nova Scotia, Canada

  2001-2005 BSc. with First Class Honours in Computer Science

Awards:

- National Sciences and Engineering Research Council of Canada Post Graduate Scholarship

  2005-2007

- A.A. Mac Donald Prize in Mathematics

  2005

- Father Ginivin Award for Excellence in Mathematics

  2003

- National Sciences and Engineering Research Council of Canada Undergraduate Student Research Award

  2001, 2002, 2003, 2004