On the Factorization of Polynomials over Algebraic Fields

submitted by

John Anthony Abbott

for the degree of Ph.D. of the

University of Bath

1988

0. Table of Contents

1. Introduction

In this introduction we describe the problem which is tackled in the thesis and our motivation for investigating this area. Having set the scene we then explain how the thesis is laid out, giving brief summaries of each chapter and showing how the chapters relate to each other.

Description of the Problem and Our Motivation

In the last ten to fifteen years there have been great advances in the power and capabilities of computer algebra systems. These systems have become more widely available as the cost of a computer powerful enough to support such system has dropped. So although the algebra systems have grown in size as increasing numbers of facilities are added to them, the computer hardware has developed faster still. Indeed, there are already home-computers with sufficient resources for the REDUCE algebra system; and another, smaller algebra system called muMATH has been available for the IBM personal computer for several years. Thus it seems likely that the number of users of algebra systems will continue to increase. The algebra systems must then respond by expanding and offering the tools that these users will need.

Certainly, the systems should be able to cope with polynomial and rational function arithmetic, at the least. Already this introduces problems: the system should to be able to remove common factors from rational functions. No user would be satisfied with an algebra system which gave an answer like

$$\frac{x^5+2x^4-4x^3+9x^2-6x+4}{x^4+2x^3-5x^2+6x-4}$$

instead of the reduced form

$$\frac{x^2-x+1}{x-1}.$$

In fact the oldest known algorithm, Euclid's algorithm, deals with exactly this problem, but it turns out that there are hidden complications. Euclid's algorithm becomes very inefficient on larger problems. Many people have tried a variety of ways of combating this inefficiency, and consequently have come up with a range of algorithms vastly superior to Euclid's. Even so, no single algorithm stands out as the "best": however, some of them do perform well in all situations. We shall see this phenomenon time and again; normally one of two attitudes is taken, either to use an algorithm that is fairly good all round, or to try to characterise the circumstances under which certain algorithms are best, and then write a program to select the best algorithm for each specific problem. Such a program is sometimes called a *polyalgorithm.*

Once the ability to manipulate rational functions has been included, we can consider further operations like differentiation and integration. Differentiation is easy. Many schoolchildren are taught how to differentiate. They are taught that

$$\frac{d}{dx}x^n = nx^{n-1}$$

and a few other rules about how to deal with products and quotients, and also the "function of a function" rule. With these rules and some standard results about trigonometric, logarithmic, and exponential functions the children can then differentiate almost anything — and some of them can even get the right answer! These rules for differentiation bear a close resemblance to a computer program, and it is not hard to write a program to differentiate anything the schoolchildren can.

Now we have an algebra system which can handle rational functions and differentiate them. How about integration? Schoolchildren find integration a bit harder.

There is an easy formula for polynomials — just differentiation in reverse. But rational functions are not so simple. Nasty surprises lurk here. The first problem comes with trying to integrate $1/x$ because the formula for integrating powers of x goes wrong. Instead of using the formula, we magically introduce a logarithm to get $\log(x)$, which differentiates back to $1/x$ so everything is fine. More difficult rational functions are tackled either by noticing that the integrand "looks similar to" one in a table so the answer can be read from the table, or by decomposing into partial fractions and then using the table. So the children's ability to integrate is restricted by the tables they use and their skill at partial fraction decomposition. Another couple of tricks they can use is substitution (often of a trigonometric function) to transform the integrand into a recognizable expression, or to apply the rule about integration by parts somehow.

We want to enable our algebra system to integrate. Computers are good at arithmetic, even on rational functions, but they are far less effective at deciding whether a formula "looks similar to" one stored in a table, or whether integration by parts can usefully be applied. Also we do not want the computer to be limited by having to look up integrals in some (finite) table.

Let us consider another way. Suppose the computer were able to factorize polynomials, e.g. $x^2-a^2 = (x+a)(x-a)$, and $x^3+2x^2+2x+1 = (x+1)(x^2+x+1)$. Then the computer could split the integrand into partial fractions whose denominators were irreducible (i.e. cannot be factorized into smaller polynomials). Those fractions with linear or quadratic denominators can easily be integrated to give a sum of logarithms and arctangents, but fractions with higher degree denominators are harder. It is well known that any polynomial in one variable can be factorized into linear factors with coefficients in $\mathbb{C}$, so if the denominator were univariate (i.e. contains just one variable), the computer could factorize it over $\mathbb{C}$ so that all the partial fractions would have linear denominators and integrate to logarithms. This is not entirely satisfactory because if the integral were just the logarithm of a quartic, say, then this method would have gone to all the trouble to factorize the quartic and produce a sum of the four logarithms of the linear factors over $\mathbb{C}$ of the quartic when a single logarithm and no factorization are all that are needed. Luckily, some more mathematics allows us to restrict our factorisation to the easiest possible one that will give the answer ([Trager76] and [Rothstein77]), e.g. in

$$\int \frac{3x^2-1}{x^3-x+1} = \log(x^3-x+1),$$

we need not factorize the cubic at all. Of course, we cannot always avoid a complete factorization, as the following example shows

$$\int \frac{dx}{x^2-2} = \frac{1}{2\sqrt{2}}\log(x-\sqrt{2})-\frac{1}{2\sqrt{2}}\log(x+\sqrt{2}).$$

This also brings us to the question of how to represent $\sqrt{2}$. We could compute an approximation to $\sqrt{2}$ by some root finding method but this is a notoriously ill-conditioned problem in general [Wilkinson59]. Alternatively, we can use an algebraic representation of the roots; i.e. we tell the computer to create a special symbol which behaves like a transcendental *except* that it yields zero when substituted into the polynomial of which it is a root (i.e. its *defining polynomial*). For example, we could create α which satisfies $\alpha^2-2=0$ to solve the integration problem above. Such symbols are called *algebraic numbers* because they represent numbers which satisfy a certain algebraic relationship. However, we find that there is a disadvantage to this approach: we may need to compute the factorization of a polynomial in terms of these algebraic numbers. Such a factorization is not always easy; for example if our algebraic number α has defining polynomial

$$\alpha^4-10\alpha^2+1=0$$

and then we determine the factorization of x^2-2 in terms of α we get:

$$x^2-2 = \left(x-\frac{\alpha^3-9\alpha}{2}\right)\left(x+\frac{\alpha^3-9\alpha}{2}\right).$$

The scheme outlined above is a viable method of integration but has the restriction that the denominator of the function to be integrated has to be univariate. Yet we can easily generalise from the example above to get

$$\int\frac{dx}{x^2-a} = \frac{1}{2\sqrt{a}}\log(x-\sqrt{a})-\frac{1}{2\sqrt{a}}\log(x+\sqrt{a}).$$

So if we allow the computer to generate symbols which behave like transcendentals except that they give zero when substituted into a multivariate polynomial (i.e. involving more one than variable) then the scheme will generalise to cover these cases. Such symbols are called *algebraic functions* since they are functions of the variables (other than the main variable) in the defining polynomial and they satisfy an algebraic relationship (viz. the defining polynomial). Unfortunately, the disadvantage cited in the previous paragraph is even more serious in this case.

The aim of this thesis is to consider ways of overcoming these disadvantages by developing efficient algorithms for producing factorizations of polynomials in terms of algebraic numbers and functions. Such algorithms are also applicable to many other areas of computer algebra like simplification of formulae, and quantifier elimination.

Thesis Organization

We have arranged the thesis into nine chapters (including this one) and six appendices mostly containing tables of results. Here we explain briefly what each chapter covers and how the chapters relate to each other.

Chapter 2 lays the mathematical foundations upon which the other chapters build. It includes all the basic definitions ranging from "algebraic number" to "integral basis". It also defines terms related to the realisation of algebraic number fields in REDUCE with some insight into the details of the implementation. We give an abstract of how the Hensel lifting technique can be applied to polynomial factorizations, and show where this fits into the "modular-Hensel" method for factorising polynomials — this being the underlying model for our algorithm. There are a few other miscellaneous definitions too.

Chapter 3 compares and discusses a wide selection of papers concerned with polynomial factorization. The papers are divided into four categories according to whether the factorization is over a finite or an infinite field and whether the polynomial to be factorized is univariate or multivariate. The discussion relies heavily on the ground work in chapter 2. There is also a short section giving an overview of papers on lattice basis reduction algorithms.

Chapter 4 abandons the introductory nature of chapters 1 to 3, and takes a close look at the relative sizes of the coefficients of polynomials and the coefficients of their factors. The results of this chapter depend on the implementation details described in chapter 2.

Chapter 5 is concerned with lattice basis reduction as in [LLL82]. In it we explain how the basis reduction is used, and give a generalisation of this to algebraic number fields with multiple generators. There is an outline of the original algorithm by Lovász which sets up notation for the chapter. We present the findings of several experiments to compare a variety of modifications to the original algorithm, and then select one as being the best overall. An empirical formula for the running time of our implementation of this algorithm is included. The chapter is virtually self-contained and uses separate notation from all the others.

Chapter 6 looks into the details of applying Hensel lifting to a factorization of a univariate polynomial. We pick four strategies from pure linear to truncated quadratic lifting, and define these using abstract algorithms. A table of results is given, and followed by simple complexity analyses which agree with the empirical data. Attention is then focused on the finer details of how to lift the factors and correction factors with several possibilities being considered. We conclude by picking the combination which appears to be most efficient in general.

Chapter 7 tackles the rather harder problem of using Hensel lifting methods on the factorizations of multivariate polynomials. It follows on from chapter 6 using the same terminology. We take a critical look at the three main papers (on the "classical" method) ignoring the more recent sparsity preserving algorithms. We also generalise Wang's leading coefficient trick to algebraic number fields.

Chapter 8 seeks an efficient way of adapting the multivariate Hensel lifting algorithm to produce factorizations in algebraic function fields. There are two alternatives: one depends upon a conjecture for its validity, the other is certain to work but is more restrictive.

Chapter 9 concludes the thesis by presenting a reasonably efficient algorithm for the factorisation of polynomials over algebraic fields based on the information discovered throughout the thesis.

Thanks and Acknowledgements

Particular thanks are owed to James Davenport for suggesting the line of research, and for showing such keen interest in my work. Special mention should be made to his amazing ability to give accurate references off the top of his head. He was also always available to correct my misapprehensions (and my mathematics), and to answer my questions. His great enthusiasm constantly spurred me on. Right from the start both he and John Fitch would willingly explain REDUCE's eccentricities as I stumbled across them. I must thank John Fitch for explaining and maintaining the Cambridge LISP upon which REDUCE and my programs ran.

Julian Padget was another useful source of help and information. His deep knowledge of Cambridge LISP frequently came in handy and led to many interesting conversations. Phil Willis was in charge of the computing group, and I thank him for organising it so well.

Russell Bradford was a perpetual source of amusement as well as an excellent colleague with whom to do research. We often worked together and criticised one another's work. He taught me to program more elegantly and to use long variable names.

Thanks also to Nick Brealey who proved a little lemma for me, and to Balbir Barn who did more than his share of the cooking and washing up! Further thanks to Erich Kaltofen for many interesting and fruitful conversations, and allowing me to finish writing this thesis at RPI. Finally, thanks to Karen, secretary to the computing group, for all her work and organisation.

2. Fundamentals and Definitions

In the introduction we explained that we chose to use a symbolic (or algebraic) representation for roots of polynomials, and we called such symbols algebraic numbers. This chapter describes some details of the algebraic number package which deals with the algebraic numbers and upon which the factoriser is built — full details are in [ABD86]. The extensions of this package to handle algebraic functions are mentioned briefly. Also some formulae and definitions related to our choice of representation are given; the formulae will be used elsewhere.

The second part of this chapter explains the modular-Hensel factorization paradigm (sometimes called the Berlekamp-Hensel paradigm) along with the algorithm of Cantor & Zassenhaus for polynomial factorization over a finite field (my explanation is only for odd characteristic). Then we give a definition of the Swinnerton-Dyer polynomials and describe their factorizations in finite fields. The last section gives a simplified description of the Hensel lifting process for polynomial factorizations, and also includes an outline of the classical (Kronecker's) algorithm.

Algebraic Number Package

We describe here the representation of elements of an algebraic number field as implemented in the Bath Algebraic Number Package (BANP). Firstly, we give some mathematical definitions needed in the description. An *algebraic number,* α, is a number which satisfies an algebraic relation; that is there is some polynomial with rational coefficients which has α as a root. It can be shown that there is a unique monic (i.e. having leading coefficient of 1) polynomial of least degree with rational coefficients which has α as a root — this polynomial is the *minimal polynomial* of α. If the minimal polynomial has integer coefficients then we say that α is an *algebraic integer.* We observe that all rational numbers are algebraic numbers since any $q \in \mathbb{Q}$ has the trivial minimal polynomial $m_q(x) := x-q$; and similarly observe that all integers are algebraic integers. Henceforth, we shall *implicitly exclude* all the rational numbers whenever we use the phrases "algebraic number" or "algebraic integer". So it is now true that the minimal polynomial of an algebraic number has degree at least two; and we define the *degree* of an algebraic number to be the degree of its minimal polynomial.

We still need a little more mathematics before we can begin. We define $\mathbb{Q}(\alpha)$, the *algebraic number field* generated by an algebraic number, α, to be all numbers which can be represented as sums of rational multiples of powers of α; in other words it is the set

$$\mathbb{Q}(\alpha) := \left\{ f(\alpha) : f(x) \in \mathbb{Q}[x] \right\}.$$

The *degree of* $\mathbb{Q}(\alpha)$ is defined to be the degree of α. In a similar way we can define the field generated by several algebraic numbers, $\alpha_1, \ldots, \alpha_n$ to be

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_n) := \left\{ f(\alpha_1, \ldots, \alpha_n) : f(x_1, \ldots, x_n) \in \mathbb{Q}[x_1, \ldots, x_n] \right\}.$$

In this thesis we shall always have an implicit ordering on the generators, so that α_i is defined as a root of its unique monic irreducible *minimal polynomial,* $m_i(x) \in \mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})[x]$; we shall say that α_i has degree (over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$) equal to the degree of m_i. By saying that m_i is *irreducible* we mean that there are no polynomials over $\mathbb{Q}(\alpha_1, \ldots, \alpha_{i-1})$ with degrees at least 1 whose product is m_i. This definition of degree allows us to define easily the degree of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ as the *product* of the degrees of the m_i. We comment that the degree of a field is well-defined, in that it does not depend on how the extension was built up: for example,

$\mathbb{Q}(\sqrt{2}, \sqrt{3}) = \mathbb{Q}(\sqrt{2} + \sqrt{3})$ and we easily see that the degree of $\mathbb{Q}(\sqrt{2}, \sqrt{3})$ is 4 which is also the degree of $\mathbb{Q}(\sqrt{2} + \sqrt{3})$ because $\sqrt{2} + \sqrt{3}$ has minimal polynomial x^4-10x^2+1.

Now we are ready to discuss how elements of an algebraic number field are represented inside BANP. Our first concern is to ensure that the representation is canonical, i.e. we do not want any element to have more than one valid representation. We begin with the simple case where the field is generated by just one algebraic integer, α. Let m_α be its minimal polynomial. Since $m_\alpha(x) \in \mathbb{Z}[x]$, we can write any polynomial $f(x) \in \mathbb{Z}[x]$ as $f(x) = q(x)m_\alpha(x) + r(x)$ where $q(x), r(x) \in \mathbb{Z}[x]$ are the quotient and remainder respectively. We note that r has degree less than that of m_α, and that r is uniquely determined by f. Upon substituting α for x in the equation above we get $f(\alpha) = q(\alpha)m_\alpha(\alpha) + r(\alpha) = r(\alpha)$ because $m_\alpha(\alpha) = 0$ by definition. This leads directly to a canonical representation: any element of $\mathbb{Q}(\alpha)$ has a unique representation as $r(\alpha)/s$ where $s \in \mathbb{Z}^+$ and $r(x) \in \mathbb{Z}[x]$ has degree less than the degree of α and the gcd of the coefficients of r is coprime to s.

This representation can be extended to fields generated by several algebraic integers, say $\alpha_1, \ldots, \alpha_n$. The representation being $R(\alpha_1, \ldots, \alpha_n)/S$ where $S \in \mathbb{Z}^+$ and $R(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ has degree in each x_i less than the degree of α_i, and the gcd of the coefficients is coprime to S.

The reader may have spotted that we insist the field generators be algebraic integers. This is not a restriction since any algebraic number can be multiplied by a non-zero integer to give an algebraic integer. The condition is purely for computational efficiency. The facility inside BANP for introducing a new algebraic number actually creates a symbol for an algebraic integer, called an *algebraic kernel,* then divides that symbol by an integer to produce the algebraic number requested. The algebraic integers used in the internal representations (i.e. α and $\alpha_1, \ldots, \alpha_n$ above) are precisely these algebraic kernels.

We point out, in passing, that the multiple generator case can be reduced to the simple case by use of *primitive elements.* In every algebraic number field (of finite degree) there is at least one algebraic integer which generates the entire field on its own; such an element is called a primitive element — unfortunately this is not always true for fields with non-zero characteristic. BANP does not use these for two reasons: the possibly lengthy calculation of a resultant is needed to find a primitive element, and they usually lead to very unnatural and cumbersome representations: e.g. $\mathbb{Q}(\sqrt{2}, \sqrt{3})$ has a primitive element $\alpha := \sqrt{2} + \sqrt{3}$ but the representation of $\sqrt{2}$ in terms of α is the rather unpleasant $\frac{1}{2}(\alpha^3 - 9\alpha)$.

More Definitions

We introduce some more phrases that are used throughout this thesis. Some phrases are widely accepted others are invented for use inside the thesis. The latter will be called "local definitions".

Integral Bases

Later on we shall see that the ring of algebraic integers in a field plays an important role. Our interest lies especially in their representations. BANP effectively uses the *obvious basis* (local definition) for $K := \mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, namely

$$basis(\alpha_1, \ldots, \alpha_n) := \{\alpha_1^{e_1} \cdots \alpha_n^{e_n} : \forall i \;\; 0 \le e_i < \partial\alpha_i\},$$

where $\partial\alpha_i$ means the degree of α_i. What this means is that any element of K is just a sum of rational multiples of basis elements — it is a $\mathbb{Q}$ vector space basis (or equivalently a $\mathbb{Q}$-basis). It would be nice if the ring of integers in K, O_K (abbreviated to O when it is clear what K is), consisted exactly of those elements of K formed by summing integer multiples of the basis elements. This is not true in general; for example, in $\mathbb{Q}(\sqrt{5})$ the element $\alpha = \frac{1}{2}(1+\sqrt{5})$ is an algebraic integer because its minimal polynomial is $m_\alpha(x) = x^2 - x + 1$. However, such $\mathbb{Z}$-bases for O do exist, and they are called *integral bases:* for example, $O_{\mathbb{Q}(\sqrt{5})}$ has $\mathbb{Z}$-basis $\{1, \frac{1}{2}(1+\sqrt{5})\}$ which is thus an integral basis for $\mathbb{Q}(\sqrt{5})$. Clearly these bases are also $\mathbb{Q}$-bases for K. BANP does not use an integral basis to represent field elements because determining an integral basis is time-consuming, and multiplication of elements thus represented is relatively inefficient.

The Defect

We have seen that the representations of algebraic integers in BANP may involve fractions, since the obvious basis is not necessarily a $\mathbb{Z}$-basis. This possibility of having fractions complicates matters a little. It is important that these fractions, in fact, have only small denominators. This subject is dealt with more fully in chapter 4 (on Bounds), but mention here the term *defect* whose definition we generalise from [Weinberger&Rothschild76]. We define the defect of a $\mathbb{Q}$-basis for K to be the largest denominator appearing in the representations of the algebraic integers in K. It immediately follows that integral bases are precisely those bases (of algebraic integers) with a defect of 1. We illustrate this: in $\mathbb{Q}(\sqrt{5})$ an integral basis is $\{1, \frac{1}{2}(1+\sqrt{5})\}$ which has defect 1, whereas the obvious basis $\{1, \sqrt{5}\}$ has defect 2.

Extension to Algebraic Functions

Most of the definitions above extend naturally to algebraic functions and algebraic function fields. We must replace $\mathbb{Z}$ by $\mathbb{Z}[z_1, \ldots, z_\tau]$ and $\mathbb{Q}$ by $\mathbb{Q}(z_1, \ldots, z_\tau)$ where $z_1, \ldots, z_\tau$ are the transcendentals occurring in one or more of the minimal polynomials; so the α_i are algebraic functions of the z_j. We retain the term *algebraic integer* for those algebraic functions whose (monic) minimal polynomials lie in $\mathbb{Z}[z_1, \ldots, z_\tau]$. The canonical representation $R(\alpha_1, \ldots, \alpha_n)/S$ has to change slightly: the numerator $R(x_1, \ldots, x_n)$ has coefficients in $\mathbb{Z}[z_1, \ldots, z_\tau]$ and the degree in each x_i is less than $\partial\alpha_i$, also $S \in \mathbb{Z}[z_1, \ldots, z_\tau]$ is coprime to the gcd of the coefficients of R. Again the algebraic integers used in the representation are algebraic kernels.

The mathematical concepts, primitive element, ring of integers, and integral basis extend. So does the notion of defect, though it is now an element of $\mathbb{Z}[z_1, \ldots, z_\tau]$. It is theoretically just as simple to compute an integral basis for an algebraic function field as it is for a number field but the calculations are much more long-winded.

Discriminant Formula

In this section we define the discriminant of a basis of an algebraic extension, we give a formula for the discriminant of the obvious basis, and a proof of the formula. The proof requires some knowledge of Galois theory. The formula has apparently been known for some time, but we believe the proof to be new.

The need for finding the discriminant of the obvious basis stems from the polynomial factoriser described in this thesis. The factoriser needs to know the defect of the basis so that it can derive the factors from an intermediate result. It seems that it is as hard to compute the defect as it is to find an integral basis (see, for example, [Bradford88]). However, the factorizer can still derive the factors if it is given a multiple of the defect, and it can be shown that the square of the defect divides the discriminant. So it is sufficient to use the discriminant in place of the defect — actually we can often find quickly the largest factor whose square divides the discriminant, and this factor clearly still suffices. This topic is treated fully in chapter 4 on coefficient bounds of factors.

Towers, Conjugates, and Norms

Before we can define the discriminant we have to bring in a few more mathematical notions to allow us to deal with the multiple generator case. We shall call the ground field upon which the algebraic extensions are built K_0. So for algebraic number fields K_0 will be $\mathbb{Q}$, and for algebraic function fields it will be $\mathbb{Q}(z_1, \ldots, z_\tau)$. We shall call the algebraic extension generators $\alpha_1, \ldots, \alpha_n$ and denote their minimal polynomials by $m_i(x) \in K_0(\alpha_1, \ldots, \alpha_{i-1})$ and their degrees by d_i. We can now write our field as $K := K_0(\alpha_1, \ldots, \alpha_n)$. It turns out to be more convenient to build up to K in a sequence of steps like this: let $K_1 := K_0(\alpha_1)$, $K_2 := K_1(\alpha_2) = K_0(\alpha_1, \alpha_2)$, etc. So we have created a *tower* of extensions by adjoining the algebraic symbols one at a time. We can display this pictorially:

$$\left.\begin{tikzcd}
K_n \arrow[r] & L_n \\
K_{n-1} \arrow[u] \arrow[r] & L_{n-1} \arrow[u, "d_n\text{ extensions}"'] \\
\vdots \arrow[u] & \cdots & \vdots \arrow[u, "d_{n-1}\text{ extensions}"'] \\
K_1 \arrow[u] \arrow[r] & L_1 \arrow[u, "d_2\text{ extensions}"'] \\
K_0 \arrow[u] \arrow[r, "id"] & K_0 = L_0 \arrow[u, "d_1\text{ extensions}"']
\end{tikzcd}\right\} \subset \overline{K}.$$

This diagram needs to be explained. The column on the left is our tower. We know from Galois theory that K_0 has a unique algebraic closure which we call $\overline{K}$. The definitions of the fields K_i are very abstract, and we have to be more specific to prove our result. Currently the field K_i is obtained from K_{i-1} by adjoining a root of m_i, but we have not said which root to adjoin. We now insist that the particular root to be adjoined be chosen, and so build up a new tower. We shall start from the same ground field, but call it L_0 this time. We construct L_1 by picking a specific root of m_1 in $\overline{K}$ and adjoining that root to L_0. In a similar fashion we construct L_2 from L_1, and so on. Clearly, the field L_i not only depends on the choice of root of m_i but also on all the earlier choices. We have d_i choices when extending L_{i-1} to L_i, hence the comments beside the right hand tower.

We call L_i an *embedding* of K_i in $\overline{K}$. By considering all the possible choices there are for extending L_i to L_{i+1} we find that K_j has $\prod_{m=1}^{j} d_m$ embeddings into $\overline{K}$. Later on it will be necessary to distinguish all these embeddings. We do this as follows. We shall write $\sigma_1, \ldots, \sigma_{d_1}$ for the different embeddings $K_1 \to \overline{K}$, and $\sigma_{11}, \ldots, \sigma_{1d_2}$ for the possible extensions of $\sigma_1 : K_1 \to \overline{K}$ to an embedding $K_2 \to \overline{K}$ and so on.

The reader may have realised that a single element of one of the K_i may have many different images in $\overline{K}$ under the different embeddings. This is nothing more than a generalisation of the fact that 2 has two square roots, namely 1.414... and −1.414... — these are just the different images of the symbol $\sqrt{2}$ in $A \subset \mathbb{C}$. We call the images of an element under the different embeddings *field conjugates.* Note that the field conjugates need not all be distinct; indeed, the field conjugates of any member of K_0 are all the same! It can be shown that the product of all the field conjugates is always an element of the ground field, K_0. This product is called the *norm* of the element (with respect to that particular extension). We shall need to use the norm maps taking elements of L_i into L_{i-1} (i.e. the product of the images under the d_i extensions of L_{i-1}); call this map N_i. We comment that norms of elements can be computed easily via resultants.

At last we can define the *discriminant* of a basis. Let the basis be $\{b_1, \ldots, b_N\}$, then its discriminant is defined as the square of the determinant of the matrix:

$$\begin{bmatrix} b_1 & b_2 & \cdots & b_N \\ b_1^{(2)} & b_2^{(2)} & \cdots & b_N^{(2)} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ b_1^{(N)} & b_2^{(N)} & \cdots & b_N^{(N)} \end{bmatrix},$$

where $b_j^{(i)}$ is the i^{th} field conjugate of b_j. In the special case that the basis is $\{1, \alpha, \ldots, \alpha^{m-1}\}$ where α has minimal polynomial m_α of degree m, the result is also called the discriminant of m_α. Additionally, in this special case, the matrix is of Vandermonde form, from which we find an alternative way of calculating the discriminant of m_α: namely $discr(m_\alpha) = resultant(m_\alpha, m_\alpha')$, where the prime denotes differentiation.

Proposition

Let K_0 be a field of characteristic zero. Let $K_i = K_{i-1}(\alpha_i)$ for $i = 1, \ldots, n$ be a tower of algebraic extensions, with the minimal polynomial of α_i over K_{i-1} being m_i of degree d_i. Further, let $N_i : K_i \to K_{i-1}$ be the norm map. Then the discriminant of the obvious basis for K_n

$$basis(K_n) := \{\alpha_1^{e_1} \cdots \alpha_n^{e_n} : \forall i \;\; 0 \le e_i < \partial\alpha_i\}$$

is

$$\mathrm{discr}(m_1)^{d_2 d_3 \cdots d_n} \times N_1(\mathrm{discr}(m_2))^{d_3 d_4 \cdots d_n} \times N_1(N_2(\mathrm{discr}(m_3)))^{d_4 d_5 \cdots d_n} \times \cdots$$

or alternatively

$$N_2 N_3 \ldots (\mathrm{discr}(m_1)) \times N_1 N_3 \ldots (\mathrm{discr}(m_2)) \times N_1 N_2 \ldots (\mathrm{discr}(m_3)) \times \cdots$$

where discr(...) denotes the polynomial discriminant function.

Proof

In the case of a simple extension the formulae above reduce to the polynomial discriminant of m_1 which is correct. So we shall concentrate on the multiple generator case. We use an iterative construction for the discriminant matrix.

$$\text{Let } D_1 = \begin{bmatrix} 1 & \sigma_1(\alpha_1) & \sigma_1(\alpha_1^2) & \cdots & \sigma_1(\alpha_1^{d_1-1}) \\ 1 & \sigma_2(\alpha_1) & \sigma_2(\alpha_1^2) & \cdots & \sigma_2(\alpha_1^{d_1-1}) \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 1 & \sigma_{d_1}(\alpha_1) & \sigma_{d_1}(\alpha_1^2) & \cdots & \sigma_{d_1}(\alpha_1^{d_1-1}) \end{bmatrix}$$

Now iteratively define for $s = 2, \ldots, n$

$$D_s = \begin{bmatrix} \delta_1(D_{s-1}) & \delta_1(\alpha_s D_{s-1}) & \cdots & \delta_1(\alpha_s^{d_s-1} D_{s-1}) \\ \delta_2(D_{s-1}) & \delta_2(\alpha_s D_{s-1}) & \cdots & \delta_2(\alpha_s^{d_s-1} D_{s-1}) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \delta_{d_s}(D_{s-1}) & \delta_{d_s}(\alpha_s D_{s-1}) & \cdots & \delta_{d_s}(\alpha_s^{d_s-1} D_{s-1}) \end{bmatrix},$$

where for clarity, we define δ_j to have the following property: $\delta_j \circ \sigma_{r_1 \cdots r_s} = \sigma_{r_1 \cdots r_s j}$ and $\delta_j \circ id = \sigma_j$ — think of δ_j as meaning pick the j^{th} possible extension of the embedding. Note that D_{s-1} is invariant under each δ_i because all the entries in D_{s-1} are fixed — the (i, j) *th* entry above could also be written as $\delta_i(\alpha_s^{j-1})D_{s-1}$.

Then for the obvious basis, the discriminant is just the square of the determinant of D_n. So it is sufficient to prove that this determinant squared is:

$$\mathrm{discr}(m_1)^{d_2 d_3 \cdots d_n} \times N_1(\mathrm{discr}(m_2))^{d_3 d_4 \cdots d_n} \times N_1(N_2(\mathrm{discr}(m_3)))^{d_4 d_5 \cdots d_n} \times \cdots .$$

We deduce the formula above. We shall inductively find a unimodular transformation which diagonalizes the matrices D_k for $k = 1, \ldots, n$ in that order. These transformations will be constructed only by row operations. The case $k = 1$ is easy; we consider the case $k > 1$. We assume we know some row operations which diagonalize D_{k-1} to give Δ_{k-1}. Applying these row operations to each block of $d_1 d_2 \cdots d_{k-1}$ rows of D_k gives:

$$\begin{bmatrix} \delta_1(\Delta_{k-1}) & \delta_1(\alpha_k \Delta_{k-1}) & \cdots & \delta_1(\alpha_k^{d_k-1}\Delta_{k-1}) \\ \delta_2(\Delta_{k-1}) & \delta_2(\alpha_k \Delta_{k-1}) & \cdots & \delta_2(\alpha_k^{d_k-1}\Delta_{k-1}) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \delta_{d_k}(\Delta_{k-1}) & \delta_{d_k}(\alpha_k \Delta_{k-1}) & \cdots & \delta_{d_k}(\alpha_k^{d_k-1}\Delta_{k-1}) \end{bmatrix} \qquad (*)$$

which has the same determinant as D_k. Again the (i, j) *th* entry of this matrix could also be written as $\delta_i(\alpha_k^{j-1})\Delta_{k-1}$.

We can find a sequence of row operations which yield a unimodular transformation sending

$$\begin{bmatrix} 1 & \delta_1(\alpha_k) & \cdots & \delta_1(\alpha_k^{d_k-1}) \\ 1 & \delta_2(\alpha_k) & \cdots & \delta_2(\alpha_k^{d_k-1}) \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 1 & \delta_{d_k}(\alpha_k) & \cdots & \delta_{d_k}(\alpha_k^{d_k-1}) \end{bmatrix} \qquad (\dagger)$$

to a diagonal matrix $diag(\lambda_1, \ldots, \lambda_{d_k})$, say. Notice that $\prod_{i=1}^{d_k} \lambda_i^2$ is equal to the square of the determinant of (†) which is just discr(m_k).

By regarding (*) as a matrix with matrix entries of the size of Δ_{k-1} (as it is shown above) we can apply the transformation which sent (†) to diagonal form to obtain the diagonal matrix: $diag(\lambda_1 \Delta_{k-1}, \ldots, \lambda_{d_k} \Delta_{k-1})$, which has squared determinant $\text{discr}(m_k)^{d_1 d_2 \cdots d_{k-1}} det(\Delta_{k-1})^{2d_k}$. The claimed result is now immediate.

The modular-Hensel Paradigm

This section contains an overview of the way the standard present-day polynomial factorization algorithms work. The main point is the diagram below.

The normal route followed during the factorization of a multivariate polynomial over $\mathbb{Z}$ is:

(i) substitute integers for all but one of the variables to get a univariate polynomial;

(ii) factorize the univariate polynomial modulo some prime p;

(iii) find a factorization of the univariate polynomial modulo p^k for some k;

(iv) deduce a factorization of the univariate polynomial over $\mathbb{Z}$;

(v) deduce a factorization of the multivariate polynomial over $\mathbb{Z}$.

There are restrictions on the permitted substitutions in (i) and on the primes allowed in (ii) which we shall not go into here. The number k can be determined from information in steps (i) and (ii) so that step (iv) will succeed. We can represent the process diagrammatically:

$$\begin{tikzcd}
F(x_1, \dots, x_n) \in \mathbb{Z}[x_1, \dots, x_n] \arrow[d, "\text{substitute}"] & & F = GH \in \mathbb{Z}[x_1, \dots, x_n] \\
f(x_1) \in \mathbb{Z}[x_1] \arrow[d, "\text{mod } p"] & & f(x_1) = g(x_1)h(x_1) \in \mathbb{Z}[x_1] \arrow[u, "\text{unsubstitute}"'] \\
f \equiv f_p(x_1) \bmod p \arrow[r, "\text{factorize}"] & f_p(x_1) = g_p(x_1)h_p(x_1) \bmod p \arrow[r, "\text{lift}"] & f \equiv f_q = g_q h_q \bmod q \arrow[u, "(\text{ mod } q)^{-1}"']
\end{tikzcd}$$

We shall refer to this diagram frequently throughout the thesis; immediately below we look at an algorithm for accomplishing the modular factorization step; and the section following that clarifies the (mod $q)^{-1}$ step.

The Cantor-Zassenhaus Algorithm

This section contains a short description of the algorithm in [CZ81] for the factorization of a polynomial over a field of odd characteristic. The algorithm begins by performing a *distinct degree factorization:* (e.g. see [Moenck77]) that is, the polynomial is split up into factors such that all the irreducible (modular) factors of the same degree occur in their own factor. That this can be done follows from the remarkable fact that:

$$x^{q^d} - x = \prod \text{all irreducible factors of degree dividing } d$$

in the field $\mathbb{F}_q$; so successive gcd computations with $x^q - x$, $x^{q^2} - x$, etc (each time dividing out the factor found) will produce in sequence the products of the irreducible factors of degree 1, then degree 2 and so on. We are now ready to give this elegant algorithm:

The Algorithm

A We may assume the polynomial, f, is square-free. Perform a distinct degree factorization to obtain a list $f_1, f_2, \cdots$ where f_d is the product of all the irreducible factors of f of degree d. For each $f_d \neq 1$ apply step **B**.

B We have been given a polynomial, $f_d \in \mathbb{F}_q$, all of whose irreducible factors have degree d; we must find these irreducible factors. If $\partial f = d$ then f is irreducible, so return. Otherwise pick a random polynomial, h, of degree $\partial f - 1$. Compute $k := gcd(f, h^{½(q^d-1)} - 1)$. Recursively apply step **B** to k and f/k.

Normally in step **B**, k is a non-trivial factor of f. The algorithm is fairly similar for fields of characteristic 2 provided that a primitive cube root of unity is known (this may require a degree 2 extension of the finite field). Very roughly, the justification behind step **B** is that each irreducible factor, g, of f_s "generates" a degree r extension of $\mathbb{F}_q$ which we may regard as $\mathbb{F}_q[x] \bmod g(x)$, and by the Chinese Remainder Theorem picking a random polynomial modulo $f(x)$ is the same as picking random polynomials modulo each of the irreducible factors simultaneously; then the $½(q^r-1)$th power of a random element of a degree r extension of $\mathbb{F}_q$ will be 1 or -1 with equal probability, hence the gcd in step **B** will pick out just those irreducible factors where the $½(q^r-1)$th power happened to be 1. So k is trivial if and only if all the $½(q^r-1)$th powers have the same value.

The Swinnerton-Dyer Polynomials

For the sake of most currently implemented polynomial factorization algorithms, it would be very handy if there were a close correspondence between a polynomial's true factorization (i.e. in the infinite field) and its factorization in some finite field. Ideally we would like to be able to pick a finite field in which the factors of the polynomial correspond directly to the true factors. Regrettably, this is not possible: the irreducible polynomial x^4+1 factorizes modulo every prime into linear and/or quadratic polynomials. [Musser78] presents a simple way of salvaging some information from factorisations in several finite fields — since the true factors map to products of factors in the finite fields, we may be able to restrict the possible degrees of the true factors. This idea still cannot show that x^4+1 is irreducible over $\mathbb{Z}$, though it shows that true factors cannot have degrees 1 or 3.

The upshot of this is that the step labeled $(\bmod q)^{-1}$ is not trivial. We know only that each true irreducible factor has image modulo q equal to a product of some collection of the irreducible factors modulo q. So to find the true factors we try all the modular factors, and remove all those leading to true factors. We then try all pairs of the remaining modular factors, then all triples and so on, always removing those modular factors found to constitute a true factor. This is guaranteed to find all the true irreducible factors but may take a long time, as we see next.

Unfortunately, the behaviour exemplified by x^4+1 is not an isolated incident: there is an infinite family of polynomials which behave similarly. Members of this family are known as *Swinnerton-Dyer* polynomials, named after their discoverer. This set of polynomials has unbounded degree yet all the members factorise into linear or quadratic factors modulo every prime. So the best that Musser's scheme could deduce is that all true factors have even degree; yet it can be shown that every member is irreducible over $\mathbb{Z}$. This feature of Swinnerton-Dyer polynomials means that they cause the standard modular-Hensel factorization algorithms to take an amount of time exponential in the degree of the input polynomial, i.e. the worst possible case.

We now characterise the Swinnerton-Dyer polynomials by defining them as a product:

$$\prod(x\pm\sqrt{p_1}\pm\sqrt{p_2}\pm\cdots\pm\sqrt{p_n})$$

where the product is taken over all possible choices of signs giving a polynomial of degree 2^n. The p_i should be square-free and multiplicatively independent, i.e. for $e_i\in\mathbb{Z}$ $p_1^{e_1}\cdots p_n^{e_n}=1$ if and only if all the e_i are zero. Another, equivalent, characterisation is that $K:=\mathbb{Q}(\sqrt{p_1},\ldots,\sqrt{p_n})$ is a degree 2^n extension of $\mathbb{Q}$, and the corresponding Swinnerton-Dyer polynomial is the minimal polynomial over $\mathbb{Q}$ of the primitive element $\theta:=\sqrt{p_1}+\cdots+\sqrt{p_n}$ which can also be written as $N_{K:\mathbb{Q}}(x-\theta)$.

This construction can be generalised to "independent" roots of any polynomials in place of the square roots. For further information see [KMS83]; [ABD85] discusses the practical importance of this generalised class of unhelpful polynomials.

Hensel Lifting

We have just seen in the scheme above that a crucial step in the modular-Hensel style algorithms is the determination of a factorization modulo p^k given a factorization modulo p. The process which achieves the determination is called *Hensel lifting* (often shortened to *lifting*). We shall give only a simplified description of the process here; a more complete presentation may be found in [Lauer83] for instance.

Lifting is used in two separate ways during factorization: one is part of the deduction of a factorization over an infinite field given only a factorization in a (suitable) finite field; the other is part of the conversion of a univariate factorization into a multivariate one. Although the underlying theory is the same for both uses, the realisation into practical algorithms is usually very different. For simplicity we shall restrict the following to the former use, and refer to [Musser75] for a clear discussion upon the latter.

We shall assume that the factorization of the monic polynomial f modulo p is $f \equiv gh$ with g and h monic and coprime to one another — we just observe that the generalisation to the case where there are many factors is not conceptually harder but it does obscure what happens by complicating the notation.

Because g and h are coprime, we can find polynomials α_g and α_h such that $\alpha_g \hat{g} + \alpha_h \hat{h} \equiv 1 \bmod p$ where $\hat{g}$ is the product of all the factors except g (i.e. h in this special case), and similarly for $\hat{h}$. This may also be viewed as $\alpha_g \equiv \hat{g}^{-1} \bmod (g, p)$ and $\alpha_h \equiv \hat{h}^{-1} \bmod (h, p)$. We call α_g the *correction factor* for g, and likewise α_h the correction factor for h.

Our immediate aim is to find monic factors G and H with $G \equiv g \bmod p$ and $H \equiv h \bmod p$ such that $f \equiv GH \bmod p^2$. We begin by writing $G = g + p\delta_g$ and $H = h + p\delta_h$, thus automatically satisfying the first two conditions. So all we need do is find δ_g and δ_h from f, g and h. The only constraints on δ_g and δ_h are that their degrees be strictly less than the degrees of g and h respectively, and that the following equation holds

$$GH \equiv gh + p(g\delta_h + h\delta_g) \equiv f \bmod p^2. \qquad (*)$$

We can rewrite this as $g\delta_h + h\delta_g \equiv (f-gh)/p \bmod p$ from which it is clear that $\delta_g := \alpha_g(f-gh)/p$ and $\delta_h := \alpha_h(f-gh)/p$ yield a solution of (*) by definition of the correction factors. However, we have no guarantee that the degrees of δ_g and δ_h satisfy the constraints, but by the form of (*) we may add an arbitrary multiple of g to δ_g and subtract the same multiple of h from δ_h. In this way we can reduce δ_g modulo g, and δ_h will simultaneously be reduced modulo h.

By the observation above $\delta_g \equiv \alpha_g(f-gh)/p \bmod (p,g)$ or more simply $\delta_g \equiv \alpha_g f/p \bmod (p,g)$. There is no need to use rational numbers because we can reduce f modulo (p^2,g) and then divide this by p without remainder. This simplification can easily be generalised to the case of many factors.

So far we have achieved only one step: from a factorization modulo p to one modulo p^2. There are two alternatives here. Either we can consider factors of the form $G + p^2\delta_G$ and $H + p^2\delta_H$ and solve for δ_G and δ_H as above to obtain a factorization modulo p^3; or we can compute new correction factors α_G and α_H which satisfy $\alpha_H G + \alpha_G H \equiv 1 \bmod p^2$ and use these instead of the previous correction factors to compute δ_G and δ_H modulo p^2 and thus go directly to a factorization modulo p^4. The first alternative is known as *linear* lifting, the second is called *quadratic* lifting. We repeat the lifting step above until the modulus has become sufficiently large — note that it does not matter whether p is prime provided that the correction factors are known.

We can see from the preceding paragraphs that linear lifting increases the exponent in the modulus one at a time, whereas quadratic lifting doubles the exponent each time. This suggests at first sight that quadratic lifting might be much faster, but it is not entirely clear because a lot more work has to be done for each quadratic step. A full comparison is the topic of chapter 6.

For the discussion above, we made the assumption that f was monic. This is not strictly necessary; all we really need to know are the true leading coefficients of the factors — this is especially relevant when lifting from a univariate factorization to a multivariate one. In the case when the factors are not monic they must be lifted so that at all times the leading coefficients of the modular factors are exactly the modular images of the true leading coefficients. This criterion uniquely determines all the factors at each lifting step, and it can be shown that this choice of correction terms is always valid. This topic is dealt with more fully in chapter 7.

The Classical Algorithm: Kronecker & Newton

This section describes the first known algorithm for factorising multivariate polynomials over $\mathbb{Z}$ — the algorithm does not use any form of Hensel lifting. It has two parts: the first is *Kronecker's trick*, the second is a univariate polynomial factorization method of which Newton was aware. These ideas have been totally ousted by vastly more efficient modern techniques, though the algorithm in [Lenstra83a] uses Kronecker's trick.

Kronecker's Trick

Our aim here is to factorize a multivariate polynomial when we have access to a black box which can factorize only univariate polynomials. Let the polynomial to be factorized be $f(x_1, \ldots, x_n)$, and let d_j be the degree of f in x_j. We shall use an inductive argument on the number of variables in the polynomial: the induction starts at 1 because the black box can factorize univariate polynomials. Now we have a polynomial with $n \geq 2$ variables and we assume that any polynomial with $n-1$ variables can be factorized. So, in particular, we can factorize $g := f(x_1, \ldots, x_{n-1}, x_{n-1}^{d_{n-1}+1})$. If we substituted $x_{n-1}^{d_{n-1}+1}$ for x_n in any factor of f we would get a product of one or more factors of g. By considering all possible products of factors of g and replacing x_{n-1}^r by $x_{n-1}^s x_n^t$ (where $r = s + td_{n-1}$ and $0 \leq s < d_{n-1}$) in the product, we generate a sequence of polynomials including all the factors of f. The factors of f can be picked out by performing polynomial divisions. Now we have the factorization of f.

Univariate Factorization (Newton)

For completeness, we now delve into the black box of the previous paragraph — this algorithm is never used as it converts a polynomial factorization problem into a much harder integer factorization problem. The box accepts a univariate polynomial with integer coefficients and computes the irreducible univariate factors over $\mathbb{Z}$. The key idea here is that a polynomial of degree n is completely determined by its values at $n+1$ different points.

Let n be the degree of the polynomial. We evaluate the polynomial at n different points $x_1, \ldots, x_n$ to get values $y_1, \ldots, y_n$. A factor of degree d will be determined by its values at $d+1$ points, say $x_1, \ldots, x_{d+1}$. Also the values of any factor at each x_j must be integer divisors of the corresponding y_j. So we just factorize all the integers y_j and try to interpolate a polynomial from all possible combinations of the divisors of $y_1, \ldots, y_{d+1}$. If we find a factor we divide it out and change the y_j appropriately. By performing the search firstly for degree 1 factors, then degree 2 and so on, we can guarantee the factors found will be irreducible.

3. Related Work

We present a survey of recent work in the area of polynomial factorization both over fields of characteristic zero and over finite fields. There are several published survey papers covering similar ground, e.g. [Kaltofen82], [Kaltofen86], [Lenstra82b] and [Davenport&Trager81]. Some papers on lattice basis reduction are reviewed too.

This chapter has five sections. The first section considers algorithms for the factorization of univariate polynomials over finite fields with emphasis on the elegant algorithm of Cantor & Zassenhaus (described in chapter 2) which is best suited to our needs. For completeness, the second section briefly surveys ways of factorising multivariate polynomials in finite fields, and explains why we do not use any of these algorithms. The third section covers algorithms for deducing a factorisation in an infinite field from one in a suitable finite field, concentrating on the Hensel lifting approach (with a potentially exponential recombination cost — see the section on Swinnerton-Dyer polynomials in chapter 2) but mentioning more recent polynomial time methods. Then the fourth section deals with schemes for deriving a multivariate factorisation from a suitable univariate one, including Hensel based methods, sparsity preserving methods, and again just mentioning the more recent polynomial time algorithms. The final section comments upon a few algorithms for finding "reduced" bases for integer lattices.

Factorization of Univariate Polynomials over Finite Fields

Berlekamp published the first efficient algorithm for factorization of univariate polynomials over finite fields in [Berlekamp67]. His algorithm is well suited to small finite fields but can become slow in larger ones, the worst case time being $O(n^3+qrn^2)$ where n is the degree of the polynomial, q is the size of the field, and $r \le n$ is the number of factors. The point to note is the linear dependency on q.

Three years later an improved version appeared in [Berlekamp70], so that factorizations were not (practically) restricted to very small fields. This newer algorithm had worst time depending on $p^{1/4}(\log p)^{3/2}$ instead of q, where p is the characteristic of the field. However, the newer algorithm involves a probabilistic search for roots of a polynomial in a finite field, and Berlekamp suggests that the expected running time should be much better than $p^{1/4+\epsilon}$. In the meantime a variant of the first algorithm was proposed in [McEliece69] where the null space basis computation was replaced by the generation of a simple sequence of polynomials. McEliece himself admits that in general his algorithm is slower than Berlekamp's but suggests that it could be faster if all the modular factors have low degree. He mentions worst case time as being $O(n^2e^{\sqrt{n}})$ but conjectures that the average time is about the same as for Berlekamp's original algorithm. However, a statistical study in [Mignotte80] shows that McEliece's algorithm does not even have polynomial average time, unlike both of Berlekamp's.

Several modifications to Berlekamp's second algorithm appeared in [Moenck77]. In particular, one used primes of a special form ($p = 2^rR+1$ with $R \approx r$), combined with a divide-and-conquer root finding algorithm, to produce the factorization deterministically in $O(n^3+n\log p(n + \log p))$ time. Sample timings in the paper showed this algorithm to be slightly slower for fields of small characteristic but much faster for large characteristic. He also discusses what effect using fast polynomial multiplication techniques has, and develops a scheme using these techniques.

After another gap of three years a new algorithm, quite different from Berlekamp's, was discovered [Rabin80]. This was a "Las Vegas" algorithm — provably correct and probably fast. Also it had a simpler structure than earlier ones. Rabin showed that the average complexity was better than for any of the others: his complexity formula assumed fast multiplication techniques for polynomials, but even with standard methods the complexity is still only $O(n^5 \log p)$ expected time.

Rabin's reign was short, for within a year [CZ81] came along. The algorithm of Cantor & Zassenhaus is a generalization of the root-finding algorithm in [Rabin80] — see the description in chapter 2. For fields of odd characteristic their algorithm is an extremely elegant, very short "Las Vegas" algorithm.

A theoretical and empirical comparison of Berlekamp's, Rabin's and Cantor & Zassenhaus's algorithms in [Calmet&Loos82] claims that Rabin's algorithm is slower than Cantor & Zassenhaus's which in turn is slower than Berlekamp's. However, their comparison was only for factorization over prime fields, and Berlekamp's algorithm was tried only for small fields. Our experiments confirm that, in general, Berlekamp's algorithm is faster than Cantor & Zassenhaus's over small prime fields, but that the situation is reversed for large prime fields (see appendix E). However, if all the factors have low degree it is more efficient to use distinct degree factorization in place of the first part of Berlekamp's method [Coppersmith&Davenport85].

We shall see later that sometimes we do not need the factors themselves, merely their degrees. This information is readily obtained from a distinct degree factorisation. Some clever mathematics in [Gunji&Amon81] enables them to deduce the same information from the dimensions of the null spaces of powers of the Frobenius homomorphism, which is particularly appropriate when Berlekamp based algorithms are being used.

An algorithm intermediate between Berlekamp's and Cantor & Zassenhaus's is presented in [Lazard82]. The algorithm is quite short; it begins in the same way as Berlekamp's, and finishes with an analogue for idempotents (polynomials equivalent to their squares modulo the polynomial to be factorized) of the Cantor & Zassenhaus probabilistic scheme. Unfortunately, Lazard also showed that using classical polynomial arithmetic, his algorithm is always inferior to Cantor & Zassenhaus's.

Factorization of Multivariate Polynomials over Finite Fields

We include this section for completeness but shall not use any of the algorithms mentioned here. There are two obvious routes to choose from when trying to factorize a multivariate polynomial over an algebraic number field:

A

$$\begin{tikzcd}
\begin{matrix}\text{multivariate}\\ \text{polynomial}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \arrow[d] & \begin{matrix}\text{multivariate}\\ \text{factors}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \\
\begin{matrix}\text{univariate}\\ \text{polynomial}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \arrow[d] & \begin{matrix}\text{univariate}\\ \text{factors}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \arrow[u] \\
\begin{matrix}\text{univariate}\\ \text{polynomial}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[r] & \begin{matrix}\text{univariate}\\ \text{factors}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[u]
\end{tikzcd}$$

B

$$\begin{tikzcd}
\begin{matrix}\text{multivariate}\\ \text{polynomial}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \arrow[d] & \begin{matrix}\text{multivariate}\\ \text{factors}\\ \text{over } \mathbb{Q}(\alpha)\end{matrix} \\
\begin{matrix}\text{multivariate}\\ \text{polynomial}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[d] & \begin{matrix}\text{multivariate}\\ \text{factors}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[u] \\
\begin{matrix}\text{univariate}\\ \text{polynomial}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[r] & \begin{matrix}\text{univariate}\\ \text{factors}\\ \text{over } \mathbb{F}_p(\beta)\end{matrix} \arrow[u]
\end{tikzcd}$$

Our reason for using **A** rather than **B** is because the extraneous factors can be (virtually) eliminated sooner and at a comparatively low computational cost. It is most important to avoid extraneous factors since they normally become dense as they are lifted, leading to large intermediate expressions, especially when the factors are multivariate as in **B.** The Hilbert Irreducibility Theorem implies that in method **A** when we reduce from a multivariate problem to a univariate one, if we pick random integer substitution values from a sufficiently great range then we can make the probability of extraneous factors appearing arbitrarily small. There is no similar result for method **B** as can be seen from this example: $x^4+10x^2y^2-y^4 \equiv (x-y)(x-2y)(x-3y)(x-4y)$ mod 5, and, in fact, this irreducible polynomial factorizes into factors of total degree at most 2 modulo all primes — this is related to the Swinnerton-Dyer polynomials.

We shall just give short comments on some of the papers in this area. In the previous section we saw that univariate polynomials may be factorized over certain finite fields deterministically in polynomial time — although the probabilistic algorithms may be faster if the field is not a prime field. Some recent results have shown that multivariate polynomials too may be factorized in polynomial time: there are both deterministic and probabilistic algorithms.

The algorithm expounded in [Lenstra83a] reduces multivariate polynomials to bivariate ones in polynomial time by Kronecker's trick of substituting high powers of one variable for another one. The bivariate polynomial is then reduced to a univariate one by substituting a suitable field element for one of the variables (i.e. working modulo a polynomial of the form $(Y-a)$). Hensel lifting to a factorization modulo $(Y-a)^k$ followed by a modified basis reduction algorithm then allows the determination of the factorization of the bivariate polynomial. For the multivariate case the same algorithm is used except that the lattice to be reduced is much larger — this directly yields the multivariate factors rather than deducing them from bivariate ones.

The algorithm in [vzG&K85a] takes a different approach, and comes in deterministic, probabilistic, and parallel versions. In it they select a variable, say x_1, to be preserved while the others undergo substitutions of the form $x_i := x_i+a_i$ chosen so that the substituted polynomial remains square-free modulo $(x_2, \ldots, x_n)$. A root of one of the irreducible factors of the univariate image is computed and lifted in a Newton/Hensel construction to become a root modulo a sufficiently high power of the ideal generated by $x_2, \ldots, x_n$. Finally, a linear system is solved to find the minimal polynomial of the lifted root (i.e. an irreducible factor of the original polynomial). Their lifting algorithm has been formulated so that it applies equally to factorization over algebraic number fields.

Factorization of Univariate Polynomials over Infinite Fields

This section concentrates on the modular-Hensel methods; the more recent papers on polynomial time algorithms are given less attention. Papers treating the cases of factorization over $\mathbb{Z}$ or over algebraic number fields are discussed.

Before 1967 the only available algorithms for factorization were the classical ones such as Newton's which transformed the problem into one of factorization of integers. But with current methods it is much easier to factorize polynomials than integers. The first step towards the current approach came when Berlekamp published an efficient algorithm [Berlekamp67] for the factorization of univariate polynomials over finite fields, which he subsequently improved [Berlekamp70]. Also, around the same time [Zassenhaus69] presented quadratic Hensel lifting, paving the way for the modern factorization algorithms. These algorithms have the following (simplified) structure:

$$\begin{tikzcd}
f(x)\in\mathbb{Z}[x] \arrow[d, "\bmod p"] & & f(x)=g(x)h(x)\in\mathbb{Z}[x] \\
f_p(x) \bmod p \arrow[r, "factorize"] & f_p(x)=g_p(x)h_p(x) \bmod p \arrow[r, "lift"] & f\equiv f_q=g_q(x)h_q(x) \bmod q \arrow[u, "(\bmod q)^{-1}"']
\end{tikzcd}$$

Probably the earliest implementation of such an algorithm was in 1971 by Musser [Musser71], and certainly by 1975 there were working implementations of such algorithms ([Musser75] and [Wang&Rothschild75]) with extensions to perform multivariate factorizations. And within a year these algorithms had been extended further so they could factorize over algebraic number fields. An apparent drawback of these algorithms is the theoretical worst case complexity which is exponential in the degree of the polynomial to be factorised: for any irreducible polynomial, f, there are infinitely many primes, p, such that f factorises completely into linear factors; so if f has degree n then during the $(\bmod q)^{-1}$ step we must try all possible combinations of up to $n/2$ factors before discovering that f is irreducible, and there are ${}^nC_1+\cdots+{}^nC_{n/2}\geq 2^{n-2}$ possibilities. However, it was widely believed that these algorithms were for all practical purposes polynomial time when operating over $\mathbb{Z}$ despite their theoretical exponential complexity, but when operating over algebraic number fields the behaviour seemed truly exponential.

The problem of producing a factorization over an algebraic number field is significantly harder. The classical reduction from factorization over an algebraic number field to factorization over the integers was improved in [Trager76] which presented a completely general algorithm capable of factorization even over algebraic function fields! However this method has some disadvantages: the transformed polynomial, to be factorized over $\mathbb{Z}$, has high degree (viz degree of original polynomial $\times$ degree of field extension) as well as large coefficients; and also the polynomials so created tend to factorize into many irreducibles in the finite field causing the undesirable exponential behaviour [ABD85].

An alternative approach described in [Weinberger&Rothschild76] is a generalization of the method for factorizing over $\mathbb{Z}$. The method computes several factorizations $\bmod p$ — one for each factor of the minimal polynomial $\bmod p$ — then combines these factorizations using the Chinese Remainder theorem in addition to the usual combinatorial search. A major disadvantage of this method is that if there are several factors of the same degree in each of the modular factorizations then the only way to find the true factors to which they correspond is to try all the possible combinations of modular images: for example, consider trying to factorize x^4-10x^2+1 over $\mathbb{Q}(\alpha)$ where α has minimal polynomial $m_\alpha(x) = x^4-24x^2+4$; if we work modulo the prime 1201 then $m_\alpha(x) \equiv (x+51)(x+259)(x+942)(x+1150)$ so we shall have to perform four factorizations over the corresponding extension fields (all are $\mathbb{F}_{1201}$ in this case); as all the finite fields are the same, each factorization will be the same, namely $f(x) \equiv (x+202)(x+327)(x+874)(x+999)$; now, to see if there is a linear factor of f we must apply the Chinese Remainder algorithm to all the $4\times4\times4\times4 = 256$ possible ways of picking a factor of f in each field corresponding to a factor of m_α; we find that there is no linear factor, so we try all the $6\times6\times6\times6 = 1296$ ways of picking pairs of factors from each of the four factorizations; in this case f turns out to be irreducible over the field given.

So if each of the factorizations $\bmod p$ contains lots of extraneous factors there will be two exponentially large searches one on top of the other — this can be arranged using two Swinnerton-Dyer polynomials: let the extension be generated by a root of a Swinnerton-Dyer polynomial, and let the polynomial to be factorized be another using different primes from those used to create the extension polynomial; so no matter which prime is used the algorithm has to consider many factorizations, one for each factor of the extension polynomial, and each factorization contains many extraneous factors of degree one or two. In this way we can find a polynomial of degree n and a degree n field extension which will cause the algorithm to perform more than n^{n^2} Chinese Remainder operations. A similar construction using the same Swinnerton-Dyer polynomial for both the extension and f gives an infinite family of examples in which none of the modular factors of f is extraneous, yet the algorithm may still need more than n^{n^2} Chinese Remainderings!

An implementation in MACSYMA showed Trager's method to be faster than a polyalgorithm [Wang76] which used Weinberger & Rothschild's if the minimal polynomial remained irreducible (i.e. not needing the Chinese Remainder algorithm) and otherwise the classical method from van der Waerden.

It was well-known that all the algorithms above had exponential worst case complexity, due the combinatorial search at the end. This behaviour was especially apparent for certain types of polynomial, e.g. those produced in Trager's method [ABD85] which are closely related to the Swinnerton-Dyer polynomials and their generalizations [KMS83]. It was not clear how to avoid this potential combinatorial explosion totally but, it could be alleviated. [Musser78] suggests that several modular factorizations be determined and used to restrict the possible degrees of factors — his model indicated that on average five modular factorizations were needed to establish irreducibility of a random input polynomial.

A year later Collins derived an important result about the average complexity of the search. Two ways of searching the products had emerged: one was to try all products of degree 1, then all products of degree 2, etc; the other was to try each modular factor, then products of pairs of modular factors, and so on. Both methods are exponential in the worst case, but [Collins79] assumes a couple of plausible conjectures and then shows that on average the latter approach takes polynomial time whereas the former is still exponential.

An important aspect of the search procedure is the trial division routine. The exponential behaviour mentioned above corresponds exactly to the case when almost all of the trial divisions must fail. So we want to detect failed trials as quickly as possible to mitigate the impact of the exponential search. We do this by utilising a sophisticated trial division technique. We observed empirically that the trial divisions which failed, produced quotients and remainders with huge coefficients; for example, trying to divide $(x-8)$ into $x^{30}-1$ will yield a quotient with coefficients greater than 10^{26} and a remainder greater than 10^{27}. Yet we can often tell just by looking at the first two or three coefficients of the quotient that they are so large that the quotient cannot be a true factor, and so the division is doomed to failure. This suggests an algorithm where the coefficients of the quotient are examined for "feasibility" as they are produced, with the trial division failing as soon as any coefficient becomes too big. We call this scheme *early abort trial division,* and it is equally applicable to trial divisions over $\mathbb{Z}$ and over algebraic number fields. Its usefulness for factorizations over algebraic function fields depends on whether reasonably small bounds on the coefficient size can be determined.

We also discovered that for trial divisions over $\mathbb{Z}$ it is usually sufficient to test the integer divisibility of the constant terms and/or the sums of the coefficients (corresponding to evaluating the polynomials at 0 or 1 and testing divisibility of the images) [ABD85]. This simple test is far less effective over algebraic number fields because of the need to divide by an algebraic number — if the field extension is large then even computing the polynomial quotient and remainder is faster.

Although the search just described is theoretically the most time-consuming stage, it was quickly observed that normally most of the factorization time was spent lifting the factors. So effort was concentrated on achieving this stage efficiently. A comparison of different schemes in [Wang79b] indicated that parallel quadratic lifting was always best, though [M&Y74] is reported to have contradictory evidence. Our experimental results in chapter 6 support Wang's view; and [Weinberger&Rothschild76] claims, without proof, that quadratic lifting is asymptotically faster by some constant factor. Zassenhaus, who wrote the seminal paper on quadratic lifting [Zassenhaus69], is convinced that quadratic lifting is faster than linear [Zassenhaus78]. The possibility of determining the factorization modulo a large prime with the intention of avoiding some lifting steps was considered in [Calmet&Loos82]. They showed that it is faster to factorize in a small field and perform more lifting steps, than it is to try to save a few of the lifting steps.

Even in the light of the research just mentioned, the lifting stage continued to dominate the other stages in terms of time consumed. However, several people had noticed that when factorizing over $\mathbb{Z}$ some of the true factors are produced correctly early in the lifting process. So it has been suggested [Wang83] that trial divisions be performed at certain points during the Hensel lifting. Of course, this can only detect true factors with small coefficients and which remain irreducible modulo the chosen prime. If any true factors are found they can be removed thus simplifying the later, more expensive, lifting steps; and possibly reducing the total number of lifting steps too. This trick is probably less effective if quadratic lifting is used because the p-adic accuracy doubles each step; though if the true factors have very small coefficients it may still be worthwhile. Also there does not appear to be a suitable generalization for factorization over algebraic number fields as the cost of computing the prospective true factor from the modular image is relatively high (in Lenstra's algorithm) if the minimal polynomials do not remain irreducible modulo p.

The next major developments in this area were the polynomial time algorithms. The first such algorithm for factorizing univariate polynomials over $\mathbb{Z}$ was published in [Zassenhaus81]. It was followed shortly in 1982 by another one presented in [LLL82].

This latter algorithm replaced the combinatorial search with Lovász's polynomial time lattice basis reduction. Unfortunately the cross-over point when the exponential time methods become slower is for much larger problems than we can currently handle — [Goebbels85] reports a modification of [LLL82] which may be faster.

Another application of Lovász's basis reduction algorithm to factorization is given in [Lenstra82] where Lenstra presents a variant of Weinberger & Rothschild's method replacing the part of the combinatorial search associated with the Chinese Remainder algorithm by a lattice basis reduction. Lenstra compared his own implementations of his and Weinberger & Rothschild's methods, and concluded that his method is superior. Lenstra's algorithm appears to be the currently fastest factorizer over algebraic number fields — certainly much faster than Trager's (see appendix F). This algorithm is a foundation stone of this thesis.

Since 1982 a lot of effort has been centred on extending the capabilities of polynomial time algorithms. The extension to factorization over algebraic number fields took less than a year: [Lenstra83] is a direct generalization of the integer case in [LLL82], but recommends [Lenstra82] for practical purposes. Some methods have been found which avoid factorization in finite fields by approximating roots: the basis reduction algorithm can be used to find minimal polynomials of rational approximations to algebraic numbers, so we can compute an approximate complex root of a polynomial and then find the irreducible factor to which it corresponds ([Lenstra84] and [Schönhage84]). Equivalently we could compute a p-adic approximation to a root and determine its minimal polynomial [Viry85]. The extensions of these algorithms to algebraic number fields would increase the dimensions of the lattices by a factor equal to the extension degree of the algebraic number field. All these algorithms are currently inferior to the modular-Hensel ones.

Factorization of Multivariate Polynomials over Infinite Fields

This section reviews some alternatives for lifting from a univariate factorisation to a multivariate one. The more recent papers about sparse lifting or lattice based schemes are commented on only briefly as these have been devised chiefly to produce polynomial time algorithms — their applicability to real problems being somewhat questionable.

All modern algorithms for factorizing multivariate polynomials work by reducing the problem to the factorization of univariate polynomial and then lifting this factorization until the true multivariate factors can be found. These Hensel based algorithms are, in general, greatly superior to the classical method (Kronecker's) of substituting high powers of one variable for all the other variables which tends to produce a univariate polynomial of extremely high degree. Another feature of the modern algorithms is that most of them perform the "easy" reductions to primitive square-free polynomials at the start; however, such calculations are significantly harder than in the univariate case (see, for example, [Wang&Trager79]).

The first of the modern algorithms appeared around 1971 [Musser71]. The more widely available paper [Musser75] presents the theory behind Hensel lifting, and then considers various possible ways of realising this. One particular interpretation is claimed to be most promising, based on a complexity analysis (not in the paper). A short while later a variant was published in [Wang&Rothschild75]. There were many similarities between them, but also some notable differences: Musser appears to be in favour of a quadratic construction for lifting the multivariate factors, and of a degree ordered search through the modular factors; whereas Wang & Rothschild prefer linear lifting and a cardinality ordered search. We have already noted that [Collins79] showed (under some plausible assumptions) the degree ordered search for univariate factors to be inferior, and it seems reasonable to expect the same for multivariate factors.

Another initial difference was that Wang & Rothschild lifted all the variables at once thus avoiding the rational function computation (and associated multivariate gcds) apparently needed in the variable-by-variable scheme favoured by Musser. In fact, Musser expressly avoided the rational functions by calculating in a polynomial ideal. However, after a couple of years Wang changed his mind [Wang77] and followed Musser's suggestions. Wang also reported that the change reduced intermediate expression size in addition to yielding greater speed.

The generalization of the lifting methods to algebraic number fields posed no real problems except that fractions may appear in the representations of the coefficients. It is quite clear from the theoretical part of [Musser75] that no further complications would arise. A comparison of [Wang76] with [Wang&Rothschild75] emphasises the similarities.

The practical limitations of the implementations of the algorithms above led to some heuristic improvements. Experience with factorizations over $\mathbb{Z}$ led to the recognition of three major problems: the leading coefficient problem, the "bad zero" problem, and the extraneous factor problem. Both the leading coefficient and the extraneous factor problem had been known from univariate factorizations, but the "bad zero" problem was new. All three problems manifested themselves in the same way, namely the formation of needlessly large intermediate expressions.

The Leading Coefficient Problem

The leading coefficient problem occurs when the polynomial to be factorized is not monic. The Hensel lifting algorithm needs to know the leading coefficients of the factors otherwise it cannot work — see chapter 2. In the univariate case we were able to force all the factors to have a leading coefficient equal to that of the original polynomial, and consequently pay a small price in having to work with slightly larger numbers than strictly necessary. Such an approach is more serious for multivariate polynomials since we generate high powers of the leading coefficient, which could be dense multivariate polynomials.

The Extraneous Factor Problem

These are factors which are not modular images of true factors, for example $x^2+6x+2y^2 \equiv (x+2)(x+4)$ mod $(y-2)$ but neither $(x+2)$ nor $(x+4)$ is an image mod $(y-2)$ of a true factor. Extraneous factors are bad news in two respects: firstly they usually become increasingly dense as the lifting proceeds, consuming lots of space and computation time; and secondly they lead to a combinatorial search at the end. Only the latter effect was apparent during univariate factorisations.

The "Bad Zero" Problem

This occurs when picking zero as a substitution value for all the variables violates one of the conditions of square-freeness and full degree of the image. Consequently, we must either calculate modulo a polynomial ideal of the form $(x_2-a_2, \ldots, x_n-a_n)$ while lifting or we must rewrite the original polynomial in terms of $y_j := x_j-a_j$. Both possibilities can produce large dense multivariate polynomials. Musser recommends rewriting the original polynomial as that happens only once at the start and once at the end whereas there may be many calculations modulo the polynomial ideal.

Some Solutions

All three problems (for factorization over $\mathbb{Z}$) were effectively tackled in [Wang78]. A clever trick solved the leading coefficient problem, a new approach precluded the "bad zero" problem, and the extra flexibility in the new algorithm allowed sufficient freedom that extraneous factors were virtually eliminated. Wang's trick for predetermining the leading coefficients is discussed in chapter 7 where a generalisation of the trick to work in algebraic number fields is given.

Another advantage of knowing the leading coefficients was that other coefficients of the factors may be determined merely by a simple division if the factors are sufficiently sparse. Wang exploited this possibility and sometimes could find the complete factorization without needing to lift. This idea of "predicting" coefficients was extended further in [Lucks86]. Lucks's method worked better than Wang's when the factors were fairly dense. Lucks stopped short of considering the largest linear system generated by the known and unknown coefficients, and solving that as far as possible — he may have decided it was not worthwhile.

The importance of preserving sparsity became widely recognised after Wang's paper appeared. A notable step in this direction was taken in [Zippel79] (also [Zippel81]). Here the usual Hensel lifting method, which we have already observed can lead to dense intermediate results, is replaced by a probabilistic lifting process. If Zippel's algorithm is lucky (which it normally is) then intermediate results are never bigger than the final result — though this may be exponentially larger than the input, e.g. consider x^p-1 (for p prime) which has size $\Omega(\log p)$ but has a factor of size $\Omega(p)$. Wang expressed some further ideas on the subject in [Wang79a], though his method cannot guarantee that intermediate expressions are no larger than the answer.

Independent confirmation of the pitfalls mentioned above is given in [Moore&Norman81] who also comment that picking suitable integer substitution values can be time-consuming. Their experience agrees with [Wang78] that two or three different integer substitutions are usually enough to avoid extraneous factors during the multivariate lifting. They adopted Zippel's ideas on preserving sparsity.

Recent Papers

There is a description in [Lugiez84] of a clever way of lifting all the variables at once. The usual method involves the computation of exponentially many derivatives:

$f - (f \bmod (x_2-a_2, \ldots, x_n-a_n)^k) =$

$$\sum_{e_2+\cdots+e_n=k} \frac{d^k f}{dx_2^{e_2} \cdots dx_n^{e_n}} (x_2-a_2)^{e_2} \cdots (x_n-a_n)^{e_n}.$$

Instead of this, Lugiez uses Euler's identity (below) about homogeneous polynomials to generate a system of k linear equations.

$$\sum_{j=1}^{n} x_j \frac{\partial f}{\partial x_j} = mf \qquad \text{Euler's Identity}$$

where f is a homogeneous polynomial of total degree m in $x_1, \ldots, x_n$. Lugiez shows that his method is better than the one in [Wang&Rothschild75] but gives no comparison with the improved version in [Wang78]. He did not consider the problem of dense intermediate results. He also mentioned the parallel with partial fraction decomposition, which is essentially the same problem. A year later he came up with a totally different lifting scheme [Lugiez85] which is currently limited to bivariate factorizations. This idea needs further development before it can compete with the more general algorithms.

Most of the recent work in this area has been more of theoretical than practical importance — none of this work is used in this thesis. These theoretical advances have aimed at producing polynomial time algorithms. As there are sparse multivariate polynomials whose factors have exponentially more terms than the original polynomial [vzG&K85b], the complexity is allowed to depend on both input and output sizes. In [vzG&K85b], there is a family of polynomials which undergo an exponential growth when made primitive or square-free — almost all of the current implementations of multivariate polynomial factorizers reduce the input polynomial to primitive square-free factors as the first two steps. The same paper gives a complete description of probabilistic algorithms for the factorization of multivariate polynomials over both finite and infinite fields in polynomial time. This paper has several notable features, such as the ability to deduce leading coefficients automatically, in addition to circumventing the need to perform square-free and primitive decompositions. Two related papers are [Kaltofen85b] which reduces a multivariate polynomial to a bivariate one in polynomial time, and [Kaltofen85a] which reverses the process (i.e. lifts the factors) in polynomial time.

A radically different angle is taken in [vdH&L85] where lattice reduction plays a key role. The idea is to replace all except one of the variables by sufficiently accurate rational approximations of algebraically independent transcendental numbers; then factorize the resulting univariate polynomial; the multivariate factors can then be derived directly by using Lovász's lattice reduction algorithm. The method has two obvious trouble spots: one is the need for simultaneous transcendence measures, the other is the slow speed of current implementations of the basis reduction algorithm. The authors suggest that the method may be useful for bivariate factorizations but probably not for polynomials in three or more variables.

Lenstra, on his own, has come up with an alternative way of using lattice basis reduction to achieve the same end [Lenstra87]. His method is similar to the standard modular-Hensel ones except that the combinatorial search is replaced by a lattice basis reduction which can be performed in polynomial time — this paper claims polynomial time but it assumes densely encoded polynomials, unlike Kaltofen's papers which assume the more realistic sparse encoding.

Lattice Basis Reduction Algorithms

This section glances at a few algorithms for finding nearly orthogonal vectors which generate the same lattice as the vectors supplied to the algorithm. An alternative equivalent viewpoint is that the algorithms find particularly short generators. There is a very pronounced trade-off between speed of achieving a reduction and the degree of the reduction produced. Lovász's algorithm in [LLL82] turns out to be the one best suited to our purpose — a few papers consider improvements to this algorithm.

The problem of finding shortest vectors, Minkowski reduced bases, etc, for integer lattices has been studied for a long time; for example, a way of finding the shortest vector was given in [Dieter75]. However, it was not until late 1981 that any polynomial time basis reduction algorithm was known. Lovász's algorithm first appeared in [LLL82].

It is a generalization of Euler's reduction method for two vectors, but the basis is only fairly weakly reduced. The reduction is nonetheless strong enough for many applications — our particular interest lies in the factorization algorithm in [Lenstra82].

Subsequent papers dealing with this topic are mostly in two categories: those which apply the algorithm ultimately to perform stronger reduction but at a cost in time complexity, for instance [Schnorr86] and [Helfrich85]; and those which endeavour to achieve the same reduction faster, such as [Schönhage84], [Schnorr85], [Kaltofen83], and [Vallée87]. Our interest centres on the latter category because we do not need an especially strong reduction, we just want an adequate reduction quickly. It is not always clear whether the "faster" algorithms are practically useful but Schönhage's improvements look promising. Afflerbach and Groethe claim to be able to compute Minkowski reduced bases very quickly [A&G85] using a clever search method.

4. Bounds

This chapter considers the question of how large the factors of a given univariate polynomial can be. Clearly the degree of any factor must be less than the degree of the original polynomial, but it is not so clear what size coefficients the factor may have. We start by giving a precise statement of the problem under investigation, and we give an example showing that the problem is not trivial (even in the simplest case of factorization over $\mathbb{Z}$). We proceed directly to a solution in two parts: firstly we go over ways of bounding the denominators that can appear, and then we show how to bound the numerators in essentially the same manner as that in [Weinberger&Rothschild76]. We state a conjecture which leads to a tighter bound for the numerator, and give supportive evidence for the conjecture.

Statement of the Problem

In this section we set up the notation for the chapter, and formulate the task exactly. Our problem is that we have a polynomial, $f(x) = \sum_{j=0}^{n} a_j x^j$, over an algebraic number field, K, and we want to know how "big" the coefficients of any factor of f can be; i.e. if $\sum_{i=0}^{m} b_i x^i$ divides $f(x)$, how "big" are the b_i? We can express this precisely.

We are given an algebraic number field K, and a polynomial f, over K. Let d be the extension degree of K over $\mathbb{Q}$, so we can choose d algebraic integers, $\beta_1, \beta_2, \ldots, \beta_d$, which form a $\mathbb{Q}$-basis for K; i.e. $K = \mathbb{Q}\langle\beta_1, \ldots, \beta_d\rangle$. We shall represent all elements of K with respect to this basis, e.g. $a_i = \sum_{j=1}^{d} a_{ij}\beta_j$ with all $a_{ij} \in \mathbb{Q}$. Let the (unknown) factor of $f(x)$ be $g(x) = \sum_{i=0}^{m} b_i x^i$. As for the a_i we represent

$b_i = \sum_{j=1}^{d} b_{ij}\beta_j$ with all $b_{ij} \in \mathbb{Q}$. Thus our aim is to bound the sizes of the denominators and numerators of the b_{ij} in terms of n, m, the a_i, and the β_j.

Motivation

We start off by demonstrating that this problem is not all that easy even in the simplest case of factorization over $\mathbb{Z}$. The intuitive first guess that factors have smaller coefficients than their product is wrong, as the following example shows:

$$x^{41}-x^{40}-x^{39}+x^{36}+x^{35}-x^{33}+x^{32}-x^{30}-x^{27}+x^{23}+x^{22}$$
$$-x^{21}-x^{20}+x^{19}+x^{18}-x^{14}-x^{11}+x^{9}-x^{8}+x^{6}+x^{5}-x^{2}-x+1$$

is divisible by the apparently much larger

$$x^{33}+7x^{32}+27x^{31}+76x^{30}+174x^{29}+343x^{28}+603x^{27}+968x^{26}+1442x^{25}$$
$$+2016x^{24}+2667x^{23}+3359x^{22}+4046x^{21}+4677x^{20}+5202x^{19}+5578x^{18}+5774x^{17}$$
$$+5774x^{16}+5578x^{15}+5202x^{14}+4677x^{13}+4046x^{12}+3359x^{11}+2667x^{10}+2016x^{9}$$
$$+1442x^{8}+968x^{7}+603x^{6}+343x^{5}+174x^{4}+76x^{3}+27x^{2}+7x+1.$$

Here we have a factor with coefficients almost 6000 times the size of the coefficients of the polynomial it divides — examples of arbitrarily high degree (and arbitrarily great coefficient growth) can be constructed using a method from [Mignotte81].

To explain why we want to know such bounds, we shall restrict to the easy case of factorization over $\mathbb{Z}$; the same arguments hold for algebraic number fields but the algebra is more complicated. We shall talk about the $(\bmod q)^{-1}$ step in the standard modular-Hensel procedure (see chapter 2 page 11). It is important that any true factor corresponding to a factor modulo q can be found quickly from the modular image. We ensure this by insisting that the modular images of coefficients of true factors be distinct — if each modular coefficient were the image of at least $k > 1$ possible true coefficients then a modular factor of degree m would have at least k^m pre-images whose coefficients permitted them to be putative true factors, and we would have to search through them all. One way to arrange for the one-one correspondence is to find a limit on the absolute value of any true coefficient and then make q greater than twice the limit (to allow for positive and negative coefficients). This is what we do, and why we need the bounds discussed in this chapter.

Bounding the Denominator

In chapter 2 the notions of algebraic integer and defect were introduced, and we shall be using them here. The aim of the first part of this section is to find a (small) multiple of the defect with minimal effort; then we consider actually computing the defect. We reiterate the argument in [Weinberger&Rothschild76]. The mainstay of what follows is (one version of):

Gauss's Lemma

Let O denote the ring of algebraic integers of our field K; and let $g(x) := \sum_{j=0}^{r} g_j x^j \in O[x]$ and $h(x) := \sum_{j=0}^{s} h_j x^j \in O[x]$ with $g_r h_s \neq 0$ then if each coefficient of $g(x)h(x)$ is divisible in O by $t \in O$ then each product $g_j h_k$ is also divisible in O by t.

Useful lemma

If $f(x) \in (1/a)O[x]$ is monic, and $f(x) = g(x)h(x)$ over K with g and h monic then $g(x), h(x) \in (1/a)O[x]$.

Proof (from [Weinberger&Rothschild76])

Pick b such that $g(x), h(x) \in (1/b)O[x]$, then $b^2 f(x) = bg(x)bh(x)$. Define g_j and h_j as in Gauss's lemma, and apply that lemma to get $(b^2/a) \mid bg_j bh_k$ in other words $ag_j h_k \in O \;\; \forall j,k$. Putting $j = r$ or $k = s$ proves that each g_j and each h_k lies in $(1/a)O$.

So, given a general polynomial $f(x) \in K[x]$, if we can find an integer, $M' \neq 0$, such that $M'f(x) \in O[x]$ then we know that the coefficients of the factors of f can be taken to lie in $M^{-1}O$ where M is the leading coefficient of $M'f(x)$. It is easy to find a suitable M using our representation: just set M' to be the lowest common multiple of the denominators appearing in the representations of the coefficients (though this value may be too large by a factor equal to the true defect) — recall that the algebraic kernels are algebraic integers so any sum of products of them is an algebraic integer.

We are now about to discover one of the complications concomitant with generalisation to algebraic number fields. In the case when $K = \mathbb{Q}$ we have the ring of integers $O = \mathbb{Z}$, but even in a simple extension, $K = \mathbb{Q}(\alpha)$, we do not necessarily have $O = \mathbb{Z}[\alpha]$. For example, if $\alpha^3-3\alpha^2-3\alpha-3=0$ then $K = \mathbb{Q}(\alpha)$ has ring of integers $O = \mathbb{Z}\langle 1, \alpha, \frac{1}{2}(\alpha^2+1)\rangle$. This means that clearing denominators by multiplying by M above does not guarantee that fractions will not appear in our representations of the coefficients of the factors. This brings us back to the defect (of a basis) which we defined in chapter 2 as

$$defect(\beta_1, \beta_2, \cdots, \beta_d) = \min\{j \in \mathbb{Z}^+ : O \subseteq j^{-1}\mathbb{Z}\langle\beta_1, \beta_2, \cdots, \beta_d\rangle\}.$$

We show that the square of the defect of a basis divides its discriminant; which means that D, the largest number whose square divides the discriminant of a basis, is a multiple of the defect. Let $\omega_1, \cdots, \omega_d$ be any integral basis, and $R = (r_{ij})$ be the matrix sending $(\omega_i)\to(\beta_i)$, i.e. $\beta_i = \sum_{j=1}^{d} r_{ij}\omega_j$. Thus by definition of an integral basis, $R \in GL_d(\mathbb{Z})$, and in particular $det(R) \in \mathbb{Z}$. It is clear from the definition of the discriminant (in chapter 2) as the square of a determinant that

$$discr(\beta_1, \ldots, \beta_d) = det(R)^2 discr(\omega_1, \ldots, \omega_d).$$

It can be shown that the discriminant of an integral basis is an integer (and is the same for all integral bases of a given field). We complete the proof by showing that the defect divides $det(R)$. Let the inverse matrix of R have entries (s_{ij}). Then by Cramer's rule $det(R)$ is a common denominator for all the s_{ij}; and, we also have that $\omega_i = \sum_{j=1}^{d} s_{ij}\beta_j$ from which it is clear that $O \subseteq D^{-1}\mathbb{Z}\langle\beta_1, \ldots, \beta_d\rangle$ **QED.**

In practice, D may be hard to find, as it apparently requires integer factorization; for example the discriminant of $x^7-11x^6-19x^5+25x^4+37x^3+18x^2+7x-19$ is 16838677302720365219 (a prime) which is square-free but we have to try all primes up to more than 1000000 to discover this (or use some sophisticated primality test). Instead of computing D we can use an easy-to-find multiple of it, such as the discriminant itself, or some intermediate compromise derived from a partial factorization — we may safely let D denote the compromise value.

Combining the results above we conclude that the largest denominator that could appear in the coefficient of any factor divides MD. However, this is often a gross overestimate even if there is no compromise in finding D; for example, in the field generated by a ninth root of 54, we calculate D to be $2^4 3^{21} = 167365651248$ whereas the smallest possible value for D is 27. This leads us to the alternative approach.

At the other end of the spectrum there is the policy of investing a great deal of effort in actually calculating the defect in the hope that it pays off by saving work in later computations. Some recent work by Bradford [Bradford88] on Zassenhaus's "second round" algorithm for determining integral bases has made this approach viable. Once we know an integral basis (and hence the defect), we can easily find the minimal value for M by expressing the coefficients of f in terms of the integral basis elements. This gives us the best possible denominator bound short of knowing the answer. We have performed some experiments to compare this approach with the less sophisticated one described at the start. Our results show that the initial investment is usually worthwhile, largely because the basis reduction becomes very much quicker with the smaller numbers. We present a selection of results in the table below:

Comparison of Denominator Bounds			
Example [Lenstra82]	Estimate [Weinberger76]	Defect	Free Defect
1	5.50	5.56	5.40
2	4.08	4.70	3.94
3	9.62	9.76	9.58
4	72.8	83.8	43.6
5	2198	1211	974

The entries in the table are total factorization times for the example indicated using the denominator bound indicated: *Estimate* means the largest number whose square divides the discriminant, *Defect* means that an integral basis was calculated for the field during the factorization and the defect was taken as the denominator bound, and *Free Defect* is the same as *Defect* but excluding the time taken to find the integral basis (if many factorizations are to be performed in the same field then the integral basis need be found only once).

Bounding the Numerator

We have achieved the first part of our aim, and now turn our attention to the second. We begin by explaining fully our goal and the route we take to reach it. Then we dive into the details of the solution.

We assume that a denominator bound has already been found, and we call it Δ. Thus with the notation at the beginning of this chapter we get $\Delta b_{ij} \in \mathbb{Z}$ for all i and j; and our goal is to get an upper bound for $|\Delta b_{ij}|$. The interest in the Δb_{ij} is because they are integers and thus (relatively) easily obtained from a modular image; so these are the values we compute in the $(\bmod q)^{-1}$ step of the factorization algorithm. We chose not to use the algorithm in [WGD82] for reasons of efficiency, even though this algorithm can derive rational numbers directly from their modular images. The results in the table immediately above discourage use of their algorithm still further.

Our route follows closely those of other authors (e.g. [Weinberger&Rothschild76]). Firstly, we find an upper bound for the magnitudes of the roots of f in $\mathbb{C}$. Then by binomial expansion we bound the magnitudes of the coefficients of any factor — any factor is merely a product of linear factors $(x-\alpha)$ as α runs through a subset of the roots of f. Finally, we deduce a bound on $|\Delta b_{ij}|$ from the magnitude bounds.

(a) Bounding Roots in $\mathbb{C}$

We are given a polynomial $f(x) \in K[x]$ and must calculate an upper bound for the magnitudes of its roots. Already there is the question of which embedding $K \to \mathbb{C}$ should we use? The bound on the roots has to be valid for all the possible embeddings. So this dissuades us from calculating the perfect bound by isolating the roots of all the embeddings to sufficient accuracy — a process which is known to be ill-conditioned anyway [Wilkinson59]. We must look elsewhere for an answer. There are several formulae in the literature which yield an upper bound, but they need to know the magnitude (in $\mathbb{C}$) of the coefficients. Although we may not know the magnitude of any image of an element of K in $\mathbb{C}$, we can still find an upper bound on the magnitudes of all possible images of that element in $\mathbb{C}$.

We consider two problems simultaneously: that of bounding the maximum of the magnitudes of the possible images of an algebraic number field element, and that of bounding the maximum of the magnitudes of the roots of a polynomial with coefficients in an algebraic number field. We begin by introducing a piece of notation: we denote the maximum of the magnitudes of the embeddings of $\alpha \in K$ into $\mathbb{C}$ by $\|\alpha\|$. It follows immediately that for any $\alpha, \beta \in K$ and $q \in \mathbb{Q}$

$$\|\alpha+\beta\| \le \|\alpha\| + \|\beta\|, \quad \|q\alpha\| = |q|\,\|\alpha\|, \text{ and } \|\alpha\beta\| \le \|\alpha\|\,\|\beta\|.$$

We argue by induction on the degree of the algebraic number field extension. If the extension degree is 1 then the field is just $\mathbb{Q}$ and it is trivial to bound the magnitude of an element of $\mathbb{Q}$. We can bound the magnitudes of the roots of $\sum_{j=0}^{n} a_j x^j \in \mathbb{Q}[x]$ just by finding the largest real root of $|a_n|x^n - \sum_{j=0}^{n-1} |a_j|\,x^j$ as described below.

Now we treat the case of an extension of degree greater than 1. We assume that any element of any field of lower extension degree can be bounded, and also the roots of any polynomial with coefficients in a field of lower extension degree can be bounded. Recall that an element of the field is represented as $\sum_{j=0}^{d_r-1} c_j \beta_r^j$ with the c_j lying in the smaller field, K_{r-1}. So we can bound all the $\| c_j \|$ by induction, and also we can bound $\| \beta_r \|$ by computing a bound for the roots of the minimal polynomial of β_r (which has coefficients in K_{r-1}). So the original field element is bounded by $\sum_{j=0}^{d_r-1} C_j A^j$ where $C_j \geq \| c_j \|$ and $A \geq \| \beta_r \|$.

It remains to bound the magnitudes of the roots of a polynomial over the field. We do this by reducing the problem directly to a root bounding problem over $\mathbb{Q}$. Let the polynomial be $\sum_{j=0}^{n} a_j x^j$ and consider the new polynomial $\| a_n \| x^n - \sum_{j=0}^{n-1} \| a_j \| x^j$. It is clear that the magnitude of any root of the original polynomial does not exceed the largest real root of the derived polynomial. One point remains: we know only upper bounds for the $\| a_j \|$ not their exact values, but this does not matter if we insist that all polynomials be monic (which they are in our application), so $\| a_n \| = 1$.

The preceding argument has left the legacy of having to find the largest real root of a polynomial of the form $\tilde{f}(x) := x^n - \sum_{j=0}^{n-1} b_j x^j$ with all the $b_j > 0$. There are three well-known theoretical bounds:

$$1+\max\{b_{n-1}, b_{n-2}, \ldots, b_0\}$$

which is due to Cauchy, and can be found in [Mignotte76];

$$\max\{nb_{n-1}, \sqrt{nb_{n-2}}, \ldots, (nb_0)^{1/n}\}$$

which is also due to Cauchy; and

$$2\max\{b_{n-1}, \sqrt{b_{n-2}}, \ldots, b_0^{1/n}\}$$

which can be found in [Knuth69] as exercise 4.6.2-20. The following lemma shows that

$\tilde{f}$ has precisely one positive root (or is just x^n).

Lemma

If $g(x) = x^n - \sum_{i=0}^{n-1} b_i x^i$ with all $b_i \geq 0$ then $g(x)$ has exactly one positive root or $g(x) = x^n$.

Proof By induction on n.
If $n = 1$ the result is trivial.
We may assume $n > 1$ and $g(x) \neq x^n$. The derivative divided by n (i.e. $g'(x)/n$) satisfies the conditions of the lemma, and so has at most one positive root. Clearly $g(x) \to \infty$ as $x \to \infty$, and $g(0)=0$. We have assumed that not all the b_i are zero, so let b_j be non-zero. Then for $\varepsilon = \frac{1}{2} b_j^{1/(n-j)}$ we have $\varepsilon^n - b_j \varepsilon^j < 0$, so in particular $g(\varepsilon) < 0$. Thus g has at least one positive root. **QED**

The three formulae above give upper bounds for the real root of $\tilde{f}$, although it is easy to compute an arbitrarily close rational approximation — which is what we do. Let us define $rb(\tilde{f})$ to be the positive root of $\tilde{f}$.

Recall that our real goal is to bound the roots of any (monic) polynomial, say $f := x^n + \sum_{j=0}^{n-1} a_j x^j$, and that $\tilde{f}$ is one of the stepping stones. We have observed experimentally that by substituting

$$x \to x - \frac{a_{n-1}}{n}$$

in f (to kill the x^{n-1} term) then computing a root bound for the substituted polynomial (by finding a close rational approximation to the positive root of $\tilde{f}$) always gives a better bound. The program used to perform the experiments is given in appendix C. Based on these results we make a conjecture:

Conjecture

Define $\tilde{f}(x) = x^n - \sum_{j=0}^{n-1} |a_j| x^j$ where $f(x) = x^n + \sum_{j=0}^{n-1} a_j x^j \in \mathbb{C}[x]$. Then, with rb defined as above,

$$|\delta| + rb(\tilde{f}_\delta) \leq rb(\tilde{f}) \qquad \forall f(x) \in \mathbb{C}[x]$$

where $f_\delta(x) = f(x-\delta)$ and δ is chosen so that f_δ has no x^{n-1} term.

(b) Binomial Expansion

The next part of our route is to derive a bound on the magnitude of the coefficients of any factor. We assume a bound, B, on the magnitude of any root of f is known. We need consider only factors of degree at most half the degree of f since at most one factor can have degree greater than half that of f.

Let us begin by considering a factor of degree r, then we can consider the effect of varying r. A factor of degree r will have coefficients bounded by $\| (x+B)^r \|_\infty$, and this bound can be attained so it is tight — $\| f \|_\infty$ means the maximum of the absolute values of the coefficients of f. By the binomial expansion

$$(x+B)^r = \sum_{j=0}^{r} x^j B^{r-j} \binom{r}{j}.$$

We want to find out which coefficient is the biggest, and to do this we look at ratio of adjacent coefficients in the expansion:

$$\frac{Coeff\,(x^{j+1})}{Coeff\,(x^j)} = \frac{B^{r-j-1}\binom{r}{j+1}}{B^{r-j}\binom{r}{j}} = \frac{r-j}{B(j+1)}.$$

By inspection we see that the ratio decreases as j increases, so the largest coefficient will be for the least integer value of j giving a ratio below 1. Thus the largest coefficient will correspond to $j = \left\lfloor \frac{r+1}{B+1} \right\rfloor$. This is the magnitude bound for a factor of degree r, and it is easy to see that a factor of higher degree has a higher bound, so we set

$r = \lfloor n/2 \rfloor$ to get a bound on the magnitudes of the coefficients of any factor of degree up to $n/2$. We could also retain r and j as parameters in the bound — see the section on trial divisions in chapter 3.

(c) Deducing a Bound on $|b_{ij}|$

We have reached the last, but hardest, leg of our journey. From **(a)** and **(b)** above, we have a denominator bound, Δ, and a magnitude bound, B, for any image in $\mathbb{C}$ of any coefficient of any factor of f. In other words for the algebraic numbers $b_i = \sum_{j=1}^{d} b_{ij}\beta_j \in K$, we know that Δb_{ij} are integers, and that all their images in $\mathbb{C}$ have magnitude at most B. Our aim is to deduce a bound on the integers $|\Delta b_{ij}|$.

The key fact is that the magnitude bound is valid for *any* image in $\mathbb{C}$. So we can set up the following system of linear equations:

$$\begin{bmatrix} \beta_1 & \beta_2 & \cdots & \beta_d \\ \beta_1^{(2)} & \beta_2^{(2)} & \cdots & \beta_d^{(2)} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \beta_1^{(d)} & \beta_2^{(d)} & \cdots & \beta_d^{(d)} \end{bmatrix} \begin{bmatrix} \Delta b_{i1} \\ \Delta b_{i2} \\ \cdot \\ \cdot \\ \Delta b_{id} \end{bmatrix} = \begin{bmatrix} \Delta b_i \\ \Delta b_i^{(2)} \\ \cdot \\ \cdot \\ \Delta b_i^{(d)} \end{bmatrix}$$

where for any $\beta \in K$ we write $\beta, \beta^{(2)}, \ldots, \beta^{(d)}$ for the field conjugates of β. Observe that all the elements on the right hand side are bounded in magnitude by ΔB. We shall derive some bounds by inverting this linear system.

Obviously the values of the b_{ij} depend on the $\mathbb{Q}$-basis $\beta_1, \ldots, \beta_d$; we shall assume that the obvious basis (defined in chapter 2) for $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_t)$ is being used:

$$\{\alpha_1^{e_1}\alpha_2^{e_2}..\alpha_r^{e_r} : 0 \le e_i < degree(\alpha_i)\}.$$

This choice of basis gives the matrix a special structure from which we can derive three formulae. By Cramer's rule each Δb_{ij} is the ratio of two determinants; the denominator being the discriminant of the basis. The formulae come from different estimates for the numerator determinant. Hadamard's bound yields:

$$|\Delta b_{ij}| < \frac{Bd^{\frac{1}{2}d}\,\|\alpha_1\|^{d(d_1-1)}\,\|\alpha_2\|^{d(d_2-1)}\cdots\|\alpha_r\|^{d(d_r-1)}}{\sqrt{discr}}$$

where d_i is the degree of α_i over the field $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_{i-1})$. If we regard the determinant as a sum of $d!$ terms and bound each term, we get

$$|\Delta b_{ij}| < \frac{Bd!\,\|\alpha_1\|^{d(d_1-1)}\,\|\alpha_2\|^{d(d_2-1)}\cdots\|\alpha_r\|^{d(d_r-1)}}{\sqrt{discr}}$$

which is never smaller than Hadamard's bound. However, a result of Landau [Mignotte74] tells us that

$$\prod\{\,|\alpha| : |\alpha|>1 \text{ and } f(\alpha)=0\} \le \|f\|_2$$

where $\|f\|_2$ denote the Euclidean norm of the coefficients of f. This inequality in conjunction with a rearrangement like $ab^2c^3 = (abc)(bc)(c)$ leads to the last formula:

$$|\Delta b_{ij}| < \frac{Bd!\,\|m_1\|_2^{\frac{d}{d_1}(d_1-1)}\,\|m_2\|_2^{\frac{d}{d_2}(d_2-1)}\cdots\|m_r\|_2^{\frac{d}{d_r}(d_r-1)}}{\sqrt{discr}}$$

where m_i is the minimal polynomial of α_i over $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_{i-1})$.

Unfortunately, a bound in Wang [Wang76] (attributed to Weinberger) which seemed to be greatly superior is erroneous [Abbott&Davenport88]. The formula for a simple extension $\mathbb{Q}(\alpha)$ of degree d was:

$$|\Delta b_{ij}| < \frac{Bd!\,\|\alpha\|^{d-1}}{\sqrt{discr}}.$$

We found a family of counter-examples to this formula. One member of the family is: $\alpha^3 = 42$ so α maps into $\mathbb{C}$ as 3.476.. or $-1.738..\pm i3.010...$ Let $w = 21\alpha^2+73\alpha-127$, so the images in $\mathbb{C}$ of w have magnitude at most $B = 381$. The formula above implies that the coefficients in the representation of w as a linear combination of $\{1, \alpha, \alpha^2\}$ are strictly less than 127 — a contradiction.

In conclusion, we have achieved all that we set out to do at the start. We are able to find the best possible denominator bound, and subject to the veracity of a conjecture we can find quite a good bound on absolute values of images in $\mathbb{C}$ of algebraic numbers. However, the results in **(c)** often yield poor bounds, though they can be shown to be tight in certain cases. The following table shows how our numerator bound compares with the "optimal" one (i.e. derived by hindsight):

Comparison of Numerator Bounds

Example	Times using our bound				Times using optimal bound			
	Bound	Lifting	Basis	Total	Bound	Lifting	Basis	Total
1	11875	2.6	0.3	8.7	191	2.0	0.2	8.1
2	4149	3.1	0.6	5.5	48	2.2	0.3	3.9
3	2208	5.9	1.3	12.3	3	2.2	0.5	5.3
4	8×10^{12}	6.7	34.1	130	400000	4.2	13.2	41
5	2×10^{28}	164	1500	4400	6×10^{13}	67.0	218	997

5. Lattice Basis Reduction

The general topic of this chapter is the basis reduction algorithm which is an essential step in Lenstra's factorizer [Lenstra82]. Although it is not assumed that the reader has already looked at [LLL82], we try to keep our notation compatible with that paper. We begin by setting up the notation and terminology for the chapter, and then give a recap of Lovász's algorithm. We follow this with a discussion of various modifications made to Lovász's lattice basis reduction algorithm as presented in [LLL82]. Our experiments had shown that the basis reduction normally consumes a significant proportion of the total factorization time, so it is important to achieve the reduction as efficiently as possible. So we summarise the results of experiments designed to compare the various modified algorithms, and then select one of these as being the "best" for our purposes. Finally, we give an empirical formula for the time taken by our chosen algorithm.

Notation and Statement of the Problem

We are given the generators $b_1, \ldots, b_n$ of a lattice in $\mathbb{Z}^n$, and the aim is to find another basis of almost orthogonal vectors; or equivalently, a basis with very short vectors: for example, (writing the basis vectors as the rows of a matrix):

$$\begin{bmatrix} 100 & 0 \\ 61 & 1 \end{bmatrix} \text{ reduces to } \begin{bmatrix} -5 & -5 \\ -12 & 8 \end{bmatrix};$$

which means that any $\mathbb{Z}$-linear combination of (100 0) & (61 1) can be written as a $\mathbb{Z}$-linear combination of (–5 –5) & (–12 8), and vice versa.

We comment that any two bases of a lattice can be mapped to one another by integral unimodular transformations; i.e. the matrices corresponding to the transformations have determinant ±1, and integer entries. We define the *orthogonality defect* of a basis to be the ratio of the product of the lengths of the basis vectors to the volume of the parallelopiped whose edges are those vectors (or, equivalently, the determinant of the matrix whose rows are those vectors, provided the matrix is square). The orthogonality defect is a measure of how far a basis is from being orthogonal — the lower the defect the closer to orthogonal the basis is (the defect is always greater then 1).

In accordance with [LLL82] we use the following notation:-

b_i are the (input) vectors defining the lattice;

b_i^* are the corresponding Gram-Schmidt vectors (i.e. the component of b_i orthogonal to

$span(b_1, \ldots, b_{i-1})$);

$B_i = |b_i^*|^2$, is the square of the Euclidean length of b_i^*;

$\mu_{ij} = (b_j^*, b_i)/B_j$ is the scaled inner product of b_j^* and b_i; and

$d_i = \prod_{j=1}^{i} B_j$.

It so happens that the input bases generated by the factorization algorithm are triangular (i.e. one vector has all except the first coordinate equal to zero, another has all except the first two, and so on) which makes the initial computation of the quantities above particularly easy.

We now recap Lovász's algorithm (using the notation above), and give an example run. Comments are between square brackets.

Lovász's Algorithm

(1) Compute the b_i^* and μ_{ij}.

$B_i := |b_i^*|^2$ and

$k := 2$.

(2) [$b_1, \ldots, b_{k-1}$ are fully reduced amongst themselves, so we look at b_k]

Subtract $\{\mu_{kk-1}\}b_{k-1}$ from b_k to ensure that $|\mu_{kk-1}| \le 1/2$.

If $B_k < (3/4-\mu_{kk-1})B_{k-1}$ then do (4) otherwise do (3) — [if swapping b_k with b_{k-1} would give a b^*_{k-1} of less than $\sqrt{3/4}$ the length of the current b^*_{k-1} then do the swap otherwise don't bother].

(3) [we add b_k to those already reduced]

Make all $|\mu_{kj}| \le 1/2$ by subtracting appropriate multiples of $b_{k-2}, b_{k-3}, \cdots$ from b_k.

If $k = n$ then exit, otherwise $k := k+1$ and go to (2).

(4) [we get a worthwhile reduction by swapping b_k with b_{k-1}]

Swap b_k with b_{k-1}, and update the corresponding μ and B values.

If $k > 2$ then $k := k-1$ [we have lost one reduced vector by swapping].

Go to (2).

We shall illustrate the algorithm by showing how the example above becomes reduced. Initially $b_1 = (100\ 0)$ and $b_2 = (61\ 1)$.

Step (1) $B_1 = 10000$, $B_2 = 10000$ and $\mu_{21} = 0.61$.

Step (2) $b_2 = (-39\ 1)$ and $\mu_{21} = -0.39$.

Step (4) $b_1 = (-39\ 1)$, $b_2 = (100\ 0)$, $B_1 = 1522$, and $\mu_{21} = -2.56$.

Step (2) $b_2 = (-17\ 3)$ and $\mu_{21} = 0.44$.

Step (4) $b_1 = (-17\ 3)$, $b_2 = (-39\ 1)$, $B_1 = 298$, and $\mu_{21} = 2.23$.

Step (2) $b_2 = (-5\ -5)$ and $\mu_{21} = 0.23$.

Step (4) $b_1 = (-5\ -5)$, $b_2 = (-17\ 3)$, $B_1 = 50$, and $\mu_{21} = 1.40$.

Step (2) $b_2 = (-12\ 8)$ and $\mu_{21} = 0.40$.

Step (3) exit.

The above example is exceptional because there were only two vectors; normally upon reaching step (3) another vector will be looked at, allowing further reductions.

Notice that even two small vectors need several vector subtractions and swaps during the reduction process.

Why do we need lattice basis reduction?

At first sight there seems to be very little connection between the factorization of polynomials and basis reduction. Indeed, Trager's algorithm [Trager76] can do everything Lenstra's [Lenstra82] can, and more; and Trager's algorithm never needs to find a reduced lattice basis. However, we have already commented that Lenstra's algorithm is significantly faster, and the work of this thesis is to extend the realm of applicability of Lenstra's algorithm to that of Trager's. Our need for basis reduction is purely for Lenstra's algorithm.

The basis reduction occurs as part of the $(\bmod q)^{-1}$ step of the overall algorithm (see diagram in chapter 2). At this stage the information we have is: a tower of extensions $K_0 = \mathbb{Q}$ and, for each $i = 1, \ldots, r$ the extension $K_i = K_{i-1}(\alpha_i)$; the minimal polynomial, m_i, over K_{i-1} of each α_i, and their corresponding modular equivalents $\overline{K}_0 := \mathbb{F}_p$ and for $i = 1, \ldots, r$ the extensions $\overline{K}_i = \overline{K}_{i-1}(\overline{\alpha}_i)$, where $\overline{\alpha}_i$ has minimal polynomial $\overline{m}_i$ over $\overline{K}_{i-1}$. Recall that m_i is square-free modulo p, and that we may have had to select the $\overline{m}_i$ from several alternatives — the basis reduction is going to compensate for the effect of discarding the other possibilities for $\overline{m}_i$.

We can use an alternative notation for the $\overline{K}_i$, namely $\overline{K}_0 = \mathbb{Z}/(p)$ and $\overline{K}_i = \overline{K}_{i-1}[x_i]/(\overline{m}_i(x_i)) = \mathbb{Z}[x_1, \ldots, x_i]/(p, \overline{m}_1(x_1), \ldots, \overline{m}_i(x_i))$. This notation allows us to express the effect of lifting more easily. Let q be some power of p. Then define R_0 to be $\mathbb{Z}/(q)$. For each i from 1 to r let $\tilde{m}_i(x) \in R_{i-1}[x]$ be the unique factor of the canonical image of $m_i(x)$ in $R_{i-1}[x]$ satisfying $\tilde{m}_i(x) \equiv \overline{m}_i(x) \bmod p$; and let R_i denote $R_{i-1}[x_i]/(\tilde{m}_i(x_i))$. Thus we obtain a succession of rings $R_0, \ldots, R_r$.

The $(\bmod q)^{-1}$ step involves taking elements of R_r to elements of K_r in a special way. We have chosen q so that each element of R_r corresponds to at most one element of K_r which is small enough to occur as a coefficient of a factor of f. We want a means of finding that element of K_r given its image in R_r. This is where the basis reduction comes into play.

To use the basis reduction algorithm we must view the coefficients of factors in K_r as elements of $\mathbb{Z}^d$. We do this in the obvious way: we already have a $\mathbb{Z}$-basis for the coefficients of any factor (once we have cleared the denominators — see chapter 4), namely the obvious basis for K_r: $\{\alpha_1^{e_1} \cdots \alpha_r^{e_r} : 0 \le e_i < d_i\}$. We merely associate the basis element $\alpha_1^{e_1} \cdots \alpha_r^{e_r}$ with the unit vector $\underline{e}_s$ where $s = e_1+d_1e_2+ \cdots +d_1d_2 \cdots d_{r-1}e_r$.

Now, R_r is a lattice inside $\mathbb{Z}^d$, generated by $q\underline{e}_1, \ldots, q\underline{e}_d$ and the vectors associated with $\alpha_1^{e_1} \cdots \alpha_r^{e_r} m_i(\alpha_i) \in K_r$ with $0 \le e_i < d_i - \partial m_i$ and for $j \ne i$ $0 \le e_j < d_j$. We can calculate a reduced basis for this lattice, say $\underline{\varepsilon}_1, \ldots, \underline{\varepsilon}_d$. It can be shown [Lenstra82] that for all values of q greater than some $q_0(B)$ the fundamental region for the $\underline{\varepsilon}_i$ contains a ball of radius B — the *fundamental region* of a basis, $\underline{v}_1, \ldots, \underline{v}_n$, is the region:

$$\left\{\sum_{i=0}^{n} \lambda_i \underline{v}_i : -\tfrac{1}{2} < \lambda_i \le \tfrac{1}{2}\right\}.$$

We assume that q has been chosen so that all coefficients of the factors of f lie inside this fundamental region.

We know that any element of $\mathbb{Z}^d$ is congruent modulo R_r to a unique element inside the fundamental region. This is how we find the "smallest" element of K_r with a given modular image.

We are now ready to present the modifications we used to try to speed up the reduction of integer lattice bases. All the versions are fairly close to the original algorithm. We describe seven variants, and then give a table comparing the performance of five of these. The comparison was based on the five examples in [Lenstra82] which we give in appendix D.

Using Rational Numbers

Our first implementation followed the diagram on page 521 of [LLL82] to the letter. All quantities were represented as rational numbers (as supplied by Cambridge LISP [Fitch77]). It soon became clear that this was hopelessly slow: the fifth factorization example in [Lenstra82] produced a basis which took more than six hours to reduce, whereas Lenstra claimed to have completed the entire factorisation in under a minute. Closer examination revealed that most of the time was spent reducing the rational numbers to minimal form, i.e. calculating integer gcds.

There was little consolation in the discovery that one of the factors Lenstra gave is reducible. Lenstra did point out that the coordinates of the input basis in his implementation were restricted to being smaller than 2^{48}, unlike ours which had coordinates as large as $7^{111} \approx 6.3 \times 10^{93}$. This emphasises the importance of using proven bounds if we wish to guarantee that the algorithm finds all of the irreducible factors.

Trying Floating Point

Our next idea was to represent the b_i^*, B_i, and μ_{ij} as floating point numbers, keeping the b_i as integer vectors. No change was made to the algorithm. Now the basis reductions were very fast but sometimes the output bases were not properly reduced. The culprit was cumulation of rounding error as the μ_{ij} were updated.

We tried to circumvent the problem by recomputing the b_i^*, but were thwarted by the need to find inner products accurately. We abandoned the use of floating point numbers because of this poor behaviour on large lattices; also there were problems with representing the very large numbers involved, and there could be portability problems.

A possibility we have yet to try is the use of high precision floating point numbers. A self-correcting algorithm is given in [Schnorr85] along with guidelines about the minimum accuracy necessary. Schnorr shows his algorithm to have superior complexity to that quoted in [LLL82], but it is not clear whether his algorithm would be faster in practice. We have not implemented Schnorr's algorithm.

Using Integers

A closer look at [LLL82] revealed that we need use only integer arithmetic: they show that the d_i are sufficient denominators for the $\mu_{ij,}$ and so we need manipulate only the d_i and $d_j\mu_{ij}$. A new version was duly implemented, and was between four and ten times as fast as the original.

However, we were still dissatisfied with the performance, and so investigated further: we noticed that intermediate calculations involved extremely large integers, which were normally the $d_j\mu_{ij}$ values associated with the last few vectors — this observation led to:

An Incremental Method

It is readily apparent that the flow of control in Lovász's algorithm is unaffected by the values of b_m,d_m and $d_j\mu_{mj}$ until k first reaches the value m. Therefore we gain nothing at all by calculating and updating the values of $d_j\mu_{ij}$ and d_i for i greater than largest value k has reached so far. In fact, since these values of $d_j\mu_{ij}$ and d_i are normally large, a lot of time is wasted manipulating them.

This suggested a sort of "lazy evaluation" scheme where each time k attains a new maximum value we immediately compute all the new $d_j\mu_{kj}$ and d_k. Thereafter, the new values are kept up-to-date no matter how small k may subsequently become — there could be a compromise here where large values of $d_j\mu_{ij}$ and d_i are "forgotten" if k becomes small.

Fortunately, the d_i and $d_j\mu_{ij}$ for the first $k-1$ vectors contain enough information to allow us to find d_k and $d_j\mu_{kj}$ without having to keep the values of the b_i^*, e.g. see algorithm R in [Kaltofen83]. Unfortunately, the relevant calculations involve summing rational numbers (to give an integer sum), and we know no *efficient* way of doing this using only integer arithmetic; but we can represent each summand as an integer and a floating point fractional part, and since the sum is an integer we can safely round the sum of the floating point parts to the nearest integer to obtain the exact result. Without such a device it is computationally expensive to find the new $d_j\mu_{kj}$ and d_k.

Looking Only at Leading Digits

In our particular application of the basis reduction algorithm, most of the original coordinates were zero or very large, with just a few 1s. The initial behaviour of the algorithm was dictated by the relative sizes of the large numbers. We guessed that the least significant digits had no influence until all the numbers were smaller, thus giving rise to another modification: we scale the basis down (effectively throwing away the least significant digits), reduce the scaled down basis and find the associated unimodular transformation, then apply the same transformation to the full-size basis with the intention of reducing its orthogonality defect preparatory to applying the reduction algorithm.

We scale down the large lattice to a smaller one by dividing all the coordinates of all the b_i by some integer, κ, and rounding. To maintain linear independence we may need to alter one of the coordinates in each vector by ± 1; in this process it is important to preserve the sign of each coordinate of each vector. Our attempts to find a way of picking a good value for κ revealed an unexpected effect.

This scheme is infeasible because the unimodular transformation matrix can have extremely large entries: e.g. when factorising the fifth example from [Lenstra82] the original basis has numbers with 104 digits, the reduced basis has numbers with 35 digits, the transformation matrix taking the reduced basis to the original basis has numbers with 69 digits (hardly surprising since the reduced basis is nearly orthogonal), but its inverse has entries with 241 digits.

We explain why this is a problem. Let L be the matrix whose rows are the basis vectors for the large lattice, and S be the matrix for the smaller (scaled down) lattice; so $L = \kappa S + \varepsilon$ where κ is the scale factor and ε is a matrix with entries not exceeding κ in absolute value. We had wanted to reduce S to the matrix R via some unimodular U (so $R = US$), in the hope that UL would have smaller orthogonality defect than L. This hope is false in the light of the observation of the previous paragraph, because $UL = \kappa US + U\varepsilon = \kappa R + U\varepsilon$, and as U can be so large $U\varepsilon$ may have entries far larger than any entry in κR or even L.

A simple way to avoid the hazard of large entries in U is to keep track of the size of the largest entry in U, and as soon as the truncation error multiplied by U (i.e. $U\varepsilon$) is large enough to affect the behaviour, we apply the transformation U to L and start again, probably with a new scale factor. We have not yet fully implemented this scheme. It is reported in [Kaltofen83] that Odlyzko has found a similar approach but using floating point numbers instead of integer quotients.

Connection with Lehmer's Integer GCD Algorithm

A definite drawback of the above approach is that part way through the reduction process those vectors that have been reduced tend to be far smaller than those not yet looked at. This means that k cannot be very large otherwise all the information in the smaller reduced vectors will be lost. We could arrange for different scale factors for each vector much as in Lehmer's algorithm for computing integer gcds [Knuth81].

The connection between basis reduction and integer gcd computation is quite close. In fact, integer gcd computation is merely basis reduction in a one-dimensional lattice: for example the lattice in $\mathbb{Z}$ generated by (169) and (481) has the reduced basis

(13) (or (−13)).

This variable scaling looks particularly promising for the case of reducing a pair of vectors. For example, consider these two bases:

$$\begin{bmatrix} 1 & -1 \\ 999 & 999 \end{bmatrix} \text{ which is reduced, and}$$

$$\begin{bmatrix} 1 & -2 \\ 999 & 999 \end{bmatrix} \text{ which reduces to } \begin{bmatrix} 1 & -2 \\ 1199 & 599 \end{bmatrix}.$$

The two original bases would appear the same if we scaled all the coordinates by 1/10 (preserving signs), but if we scale the short vectors by 1 and the long vectors by 1/10 then we get a smaller basis whose transformation to a reduced basis yields the correct reduction for the original basis. We applied this idea to the version which performs "localized" reduction of pairs of vectors — for details see page 5.12.

Preprocessing the Basis

A rather different approach was to preprocess the basis to reduce its orthogonality defect by a few quick and simple transformations before using the full power of Lovász's algorithm. Just by looking at the original bases one can see many "obvious" reductions. This led to the idea of writing a routine to simulate crudely the reduction algorithm.

In essence the algorithm pretends that $b_i^* = b_i$, hoping that normally this would not be too inaccurate. In detail, the preprocessing algorithm is as follows:

(1) Set $k = 1$.

(2) We assume $b_1, \ldots, b_{k-1}$ are already $\mathbb{Z}$-reduced ($k \geq 2$ always). We ensure all $\mu'_{kj} := (b_k, b_j)/|b_k|^2 \in (-\frac{1}{2}, \frac{1}{2}]$ by subtracting multiples of $b_{k-1}, b_{k-2}, \cdots$ from b_k repeatedly until the condition is met.

(3) If $|b_k| \geq |b_{k-1}|$ we just increase k by 1 or exit if there are no more vectors. Otherwise we swap b_k and b_{k-1}, reduce k by 1 (unless $k = 2$) and go to step (2).

By saying that $b_1, \ldots, b_{k-1}$ are $\mathbb{Z}$-reduced, we mean that $|b_i| \le |b_{i+1}|$ for $i = 1, \ldots, k-2$ and that $|\mu'_{rs}| \le ½$ for all $0 < s < r < k$.

A crude analysis of this algorithm yielded an atrocious worst case complexity, due to the formulation of the test in step (3). Nevertheless it processed the bases produced during the factorisation of the five examples in [Lenstra82] about thirty times as quickly as the original routine, and the final orthogonality defects were only slightly greater than those of the fully reduced bases.

The bad news is that sometimes step (2) can be very slow, the reason being that ensuring that all $\mu'_{kj} \in (-½, ½]$ is not entirely trivial. Since the b_i are not actually orthogonal, some μ'_{kl} may change value when a multiple of b_j is added to b_k. In particular the value of μ'_{kl} is no longer guaranteed to lie in $(-½, ½]$ and may lead to a re-reduction. Just occasionally, the program has to go back and forth many thousands of times as various μ'_{kl} wander outside $(-½, ½]$ before control can pass to step (3).

We tried a crude method of solving this problem: we used the incremental basis reduction routine to complete the reduction when more than, say, 100 attempts at taming the μ'_{kj} had occurred. This hybrid showed good overall performance although the time taken varied erratically with the size of the input basis.

Localized Reduction of Blocks

We noticed that the program often swapped (step (4) in Lovász's algorithm) the same pair of vectors several times in succession. And after each swap the relevant μ_{ij} must be updated, involving a lot of needless computation. We can achieve this more efficiently by reducing the pair of vectors completely (only updating μ_{kk-1}), and then updating the other μ_{ij} and d_i just once at the end. The benefits were instant: incorporating this idea into the incremental algorithm produced a routine about as fast as the hybrid preprocessing version at its best, and which showed no signs of erratic variation. This is currently the best version.

[Schönhage84] discusses a similar approach allowing block reduction of several vectors before updating all the μ_{ij}. He claims superior asymptotic complexity over Lovász's algorithm for the correct choice of block size. We have not implemented Schönhage's algorithm.

We had hoped to observe a distinct improvement in this version when we modified it to use a variable scaling scheme. Indeed experiments showed that the part of the algorithm which had consumed most of the time was greatly accelerated by the change. However, although we found the correct linear combination much faster, all the time that was saved in that section was consumed by the scaling down computations and the application of the transformation to the two full-size vectors.

Comparison of the Algorithms

Below is a table of times taken by five of the versions mentioned above. The headings *Rational, Integer, Incremental, Preprocess,* and *Block* refer respectively to the original implementation using LISP rational numbers, the version using the d_i as denominators, the incremental version, the preprocessor assisted by the incremental routine, and the version which uses blocks of two vectors. The bases used for testing all come from trying to factorise Lenstra's five examples (using Weinberger's estimate for the denominators); the sixth basis is an alternative basis produced from Lenstra's fifth example. The examples are given in appendix D.

Comparison of Basis Reduction Routines

Basis	Time taken for the Reduction (seconds)				
	Rational	Integer	Incremental	Preprocess	Block
1	2.96	0.30	0.32	0.22	0.30
2	13.14	1.46	1.16	0.80	0.78
3	63.38	5.02	3.56	2.52	3.08
4	2356	248	107	27.02	55.22
5	18714	5428	2801	2291	1449
6	23356	4881	2362	1214	1423

From this table we see that both *preprocess* and *block* display superior performance for our type of problem. The tabulated results also hint at the large variations observed for the *preprocess* method. We select *block* as being best suited for our purposes because of its consistency.

Empirical Complexity Formula

One problem we identified early on during the implementation of the factoriser was what criterion should we use to guide our choice(s) of finite field. Experiments showed that picking different finite fields could cause a great variation in the overall factorisation time. It soon became clear that choices favourable for the Cantor-Zassenhaus factorizer were unfavourable for the lattice basis reduction, and conversely. We decided to try to estimate the total running times parameterised by the choice of finite field (so all the minimal polynomials of the extension generators must be factorised) and the distinct degree factorization of f. This spurred us to investigate the dependency of the basis reduction on these parameters. We give our conclusions below.

Extensive timings using the *block* modification (on various bases produced during factorisations over simple extensions of $\mathbb{Q}$) have yielded an empirical formula for the complexity: $time \approx (n^7\delta^5 d^{10})^{1/3}$ where $n = \log(\mathit{Hensel\ bound})$ which is the same as the maximum length of any coordinate of any (input) basis vector, δ is the finite field extension degree (so there were δ orthogonal vectors in the input basis), and d is the dimension of the lattice (i.e. the original extension degree). The formula was derived by a trivariate linear regression of $\log(time)$ against $\log(n)$, $\log(d)$, and $\log(\delta)$ [using *Minitab*]. The times were obtained for $n = 1, \ldots, 20$, $d = 4,5,6,7$, and $\delta = 1, \ldots, d-1$: we picked a random polynomial of degree d with small coefficients (<100) and having full Galois group; then different choices of prime and modular factor gave the various possible δ values; finally, the bases were generated by lifting linearly to p-adic height just greater than $(2^{24})^n$ — the choice of 2^{24} was based on the mistaken belief that the underlying large integer arithmetic routines worked with 24 bit “digits”; actually the routines used 32 bit “digits” but this should introduce only a constant scale factor. The multiplication routine used a classical algorithm as opposed to a more modern “fast” method.

A series of experiments addressed the question of where in the lattice basis reduction routine does the time go. The results were slightly unexpected: normally most of the time is spent computing the linear combinations of b_{k-1} and b_k (about 40% of the total reduction time), then second and third places went to updating the values of the μ_{ij} according to the linear combinations computed, and to the test whether to swap (roughly 20% each, but there was considerable variation). Of the rest of the time, about half was spent finding the initial values of μ_{ij} when the vector b_i is first included in the basis.

6. Univariate Hensel Lifting

This chapter presents our experiences with several ways of lifting a factorization of a univariate polynomial mod p to a factorization mod p^k. We start by stating our basic assumptions and defining the quantities used in the algorithms. Then the four lifting schemes are explained in detail, with a comparison of all four, including experimental results which support our choice of truncated quadratic lifting as being the best general purpose scheme. The last part looks closely at the implementation considerations for lifting the factors and correction factors.

Assumptions

We begin by outlining the assumptions for the whole chapter. We consider only the case where the factors to be lifted are coprime — this is automatically true if the original polynomial is square-free. The schemes we discuss below all lift the factors in parallel as suggested in [Wang76] — [Musser75] explicitly indicates use of a serial approach which simplifies notation and programming, and Zassenhaus's papers ([Zassenhaus69] and [Zassenhaus78]) imply the same though possibly only for notational clarity. We view the problem as that of lifting *factors* (as do Musser and Wang) as opposed to lifting primitive idempotents which Zassenhaus discusses.

Notation

We denote the polynomial to be factorized by f. Let p be a prime modulo which f is square-free (and of full degree) and let the factors of f mod p be $f_1, \ldots, f_s$ — in fact, they will be irreducible in our application, but we need only know that they are coprime in this chapter. For clarity we shall assume that f and all the f_i are monic. All the algorithms use what we call the *correction factors,* written as $\alpha_1, \ldots, \alpha_s$ to produce the refined factorization. The correction factors are defined as the reciprocals of the products of all the other factors in the following sense: $\forall j \ \ \alpha_j \hat{f}_j \equiv 1 \bmod (p, f_j)$, or equivalently $\alpha_j := (\hat{f}_j)^{-1} \bmod (p, f_j)$ where $\hat{f}_j := \prod_{i \neq j} f_i$. We are ready to give definitions of the algorithms and pass a few comments on them. The algorithms below employ the simplification to the lifting of the factors outlined in chapter 2 and examined later in this chapter (page 6.9).

Pure Linear Lifting

This is the simplest scheme. A factorization $\bmod p$ is refined to a factorization $\bmod p^2$, and then to one $\bmod p^3$, and so on. The algorithm looks like:

```
Input:   Factors $f_1, \ldots, f_s \bmod p$ of the univariate polynomial $f$, and the desired
         degree of refinement $p^k$.
Output:  Refined factors of $f$: $f_1, \ldots, f_s \bmod p^k$.
(1)      $q := p$.
(2)      Compute the correction factors $\alpha_1, \ldots, \alpha_s \bmod q$.
(3)      While $q < p^k$ do
(3.1)    $q := q \times p$.
(3.2)    Lift each factor: for $j = 1, \ldots, s$ do $f_j := f_j + (\alpha_j f) \bmod (q, f_j)$.
```

It is clear that this algorithm needs k steps to compute the answer. However, each step is very easy.

Pure Quadratic Lifting

This scheme is nearly as simple as pure linear lifting, the essential difference being that the correction factors are computed inside the loop. The name derives from the fact that a factorization mod p^{2^j} is lifted to one mod $p^{2^{j+1}}$ each time round the loop. The algorithm for quadratic lifting is:

```
Input:  Factors $f_1, \ldots, f_s$ mod $p$ of the univariate polynomial $f$, and the desired
        degree of refinement $p^k$.
Output: Refined factors of $f$: $f_1, \ldots, f_s$ mod $p^K$ where $K$ is the least power of 2
        greater than or equal to $k$.
(1)     $q := p$.
(2)     While $q < p^k$ do
(2.1)   Compute the correction factors $\alpha_1, \ldots, \alpha_s$ mod $q$ — not actually needed on
        the final iteration.
(2.2)   $q := q^2$.
(2.3)   Lift each factor: for $j = 1, \ldots, s$ do $f_j := f_j + (\alpha_j f)$ mod $(q, f_j)$.
```

It is easy to see that the loop is executed at most $1+\log_2(k)$ times, but more work has to be done each time round the loop. Another drawback of quadratic lifting is that p^K may be much larger than p^k — in fact, p^K may be almost as large as $(p^k)^2$. The disadvantage of this *overshoot* is that the computations at needlessly high accuracy are time consuming. Indeed, if classical arithmetic is used then the last time round the loop will take about four times as long as the penultimate time (assuming all polynomials are completely dense), and about sixteen times as long as the antepenultimate one, etc. because the computation time is dominated by the cost of the integer multiplications. Thus about three-quarters of the total lifting time is taken in the final iteration of the loop; hence, in the worst case, virtually three-quarters of the lifting time is wasted.

Fast Linear Lifting

We observed earlier that pure linear lifting requires many iterations. Although the computations inside each loop are quick and simple, they are not quick enough to compensate for the large number of iterations. The method used in REDUCE's factorizer endeavours to alleviate this: both the factors and the correction factors are lifted quadratically until the modulus is just smaller than the wordsize of the computer (assuming that single word integer computations are significantly faster than multiword ones), thereafter only the factors are lifted (necessarily linearly).

The benefits of this scheme are that fewer iterations are needed than for pure linear lifting, yet each iteration is almost as quick and easy (except for the first few negligible quadratic steps). There may be a small amount of overshoot but this is regarded as being small enough not to matter. Here is the algorithm:

```
Input:   Factors $f_1, \ldots, f_s$ mod $p$ of the univariate polynomial $f$, and the desired
         degree of refinement $p^k$.
Output:  Refined factors of $f$: $f_1, \ldots, f_s$ mod $p^K$ where $K \geq k$ is at most wordsize
         times $k$.
(1)      $q := p$.
(2)      While $q < p^k$ do
(2.1)    If $q <$ wordsize then $Q := q$ and compute the correction factors
         $\alpha_1, \ldots, \alpha_r$ mod $q$. [quadratic step only if $q <$ wordsize]
(2.2)    $q := q \times Q$.
(2.3)    Lift each factor: for $j = 1, \ldots, s$ do $f_j := f_j + (\alpha_j f)$ mod $(q, f_j)$.
```

We can see a close similarity between this algorithm and that for pure quadratic lifting — they are the same until the condition in (2.1) becomes false when the algorithm looks just like the linear lifting one.

Truncated Quadratic Lifting

Our experiments showed that pure quadratic lifting could be very fast, but also that it performed badly when it produced a large overshoot. Clearly a way of avoiding excessive overshoot would improve the poor performance in those cases. Truncated quadratic lifting was designed to fulfill this purpose. We lift quadratically while the desired accuracy is greater than the fifth power or if it happens to lie between the third and fourth powers. Then we perform one, two or four linear lifting steps (i.e. correction factors are not lifted) to pass the desired accuracy. We explain later (page 6.8) how this criterion arose. This scheme will be the same as pure quadratic lifting except in the cases where particularly great overshoot would occur. A suitable algorithm is:

```
Input:   Factors $f_1, \dots, f_s$ mod $p$ of the univariate polynomial $f$, and the desired
         degree of refinement $p^k$.
Output:  Refined factors of $f$: $f_1, \dots, f_s$ mod $p^K$ where $3k/2 > K \geq k$.
(1)      $q := p$.
(2)      While $q < p^k$ do
(2.1)    If $q^5 < p^k$ or $q^3 < p^k \leq q^4$ then compute the correction factors
         $\alpha_1, \dots, \alpha_s$ mod $q$ and $Q := q$.
(2.2)    $q := q \times Q$.
(2.3)    Lift each factor: for $j = 1, \dots, s$ do $f_j := f_j + (\alpha_j f)$ mod $(q, f_j)$.
```

Notice how similar this is to the pure quadratic and fast linear cases, the only change is the condition at (2.1). We explain below why we chose the condition at step (2.1).

Comparison of the Algorithms

We implemented all four of the algorithms above and tried them on many different examples. The table below gives the times (in seconds) taken to perform a few Hensel lifts. The examples 1 to 5 are taken from [Lenstra82] (see also appendix D), the last example was produced in the course of factorizing x^9-54 over an extension of $\mathbb{Q}$ by one of its roots. The dominance of truncated quadratic lifting over the alternatives considered here is plain to see. The figures also highlight how much time can be wasted [difference between pure quadratic and truncated quadratic] during the final iteration in pure quadratic lifting even though our implementation of the pure quadratic algorithm did not lift the correction factors on the last iteration.

Comparison of Hensel Lifting Methods

Example	Lifting Method			
[Lenstra82]	Pure Linear	Fast Linear	Pure Quadratic	Truncated Quadratic
1	5.78	2.10	1.88	1.90
2	4.48	1.94	2.48	1.94
3	14.58	4.44	5.50	4.30
4	40.94	7.50	4.82	4.82
5	1786	246	106	106
x^9-54	1201	169	66.70	43.70

We give here a theoretical argument to support our selection of algorithm. Our argument works by estimating the relative times of the different algorithms. We shall be realistic and suppose that classical arithmetic is used, so that the cost of multiplying two integers together is proportional to the product of their lengths; and in particular, arithmetic mod p^j takes time proportional to j^2. We also make a couple of plausible assumptions: the first is that all polynomials involved in the lifting process are completely dense, so that the cost of each lifting step depends only on the initial and final moduli (in fact, we take the cost to be proportional to the square of the final modulus). The second assumption is that the cost of lifting the correction factors (in quadratic steps) is proportional to the cost of lifting the factors with proportionality constant κ which depends on the degrees of the factors but is independent of the modulus.

The analysis for pure linear lifting is easy. The total cost is just the sum of the costs of lifting the factors alone by one power of p each step. Hence the entire cost is roughly proportional to

$$\begin{aligned}\text{Pure Linear} &= 4 + 9 + 16 + \cdots + k^2\\ &= (2k^3+3k^2+k-6)/6\end{aligned}$$

The fast linear algorithm is the same except for a practically negligible contribution from the initial quadratic phase. Suppose the quadratic phase lifted the correction factors to modulus p^j, then the overall cost is:

$$\begin{aligned}\text{Fast Linear} &= \textit{quadratic part} + j^2(4 + 9 + \cdots + K^2)\\ &= \textit{quadratic part} + j^2(2K^3+3K^2+K-6)/6\end{aligned}$$

where $K := \left\lceil \frac{k}{j} \right\rceil$ is the number of linear steps taken by the algorithm. If we ignore the contribution from the initial quadratic lifting then we see that the cost is roughly $k^3/3j$ which is about $1/j^{th}$ of the asymptotic cost of pure linear lifting. Allowing for the fact that we neglected the cost of the quadratic lifting this suggests we should take j to be as large as possible.

For quadratic lifting we must include the cost of lifting the correction factors. Let K be the least power of 2 greater than or equal to k. So we shall lift both correction factors and factors to modulus $p^{K/2}$ then just the factors to modulus p^K. The cost for lifting the factors is thus $4+16+\cdots+4^K$; and for the correction factors it is $\kappa(4 + 16 + \cdots + 4^{K-1})$. Summing the geometric series we deduce the total cost as:

$$\text{Pure Quadratic} = 4(4^K+\kappa 4^{K-1}-1-\kappa)/3.$$

In the worst case 2^K is almost $2k$, and substituting $2k$ for 2^K into the total cost we conclude that the cost is at most $4(4k^2+\kappa k^2-1-\kappa)/3$. This is obviously quadratic in k whereas the complexity for either of the linear algorithms is cubic. Further experiments showed that for very small values of k, fast linear lifting is better than pure quadratic, but in practice k is never that small.

The basic idea behind truncated quadratic lifting is to lift quadratically until the last few lifts when it becomes more efficient to lift linearly. However, we must be precise about when to switch to linear lifting. Using the assumptions above we can estimate the cost of various lifting combinations, and compare these:

Costs of Lifting Combinations		
Combination	Final modulus	Cost
L	p^2	4
LL	p^3	13
LLL	p^4	29
LLLL	p^5	54
LLLLL	p^6	90
LLLLLL	p^7	139
LLLLLLL	p^8	203
LLLLLLLL	p^9	284
QL	p^4	$20+4\kappa$
QLL	p^6	$56+4\kappa$
QLLL	p^8	$120+4\kappa$
QLLLL	p^{10}	$220+4\kappa$
QQL	p^8	$84+20\kappa$
QQLL	p^{12}	$228+20\kappa$
QQQL	p^{16}	$340+84\kappa$

The unit of cost is a quarter of the cost of lifting from modulo p to modulo p^2.

By considering this table we see that if we need to lift beyond p^5 (starting from p) then it always more efficient to perform a quadratic step; however, for smaller lifts linear lifting is superior except if we have to lift to p^4 when a quadratic step is worthwhile provided $\kappa < 2$. The table also shows how much can be saved by using truncated quadratic lifting versus pure quadratic or pure linear lifting: for example, to lift to p^{10} costs respectively $220+4\kappa$, $340+84\kappa$, and 384 — so we win unless $\kappa > 41$. Using the lifting algorithms detailed in the rest of this chapter, we have found experimentally that κ is usually in the range $0.9 < \kappa < 1.4$. Finally, we observe that by its very construction truncated quadratic lifting is never inferior to pure quadratic lifting.

Lifting the Factors

Here we state the generalised form of Hensel lifting referred to in chapter 2 (page 13). We have available a factorization modulo p^a, say $f \equiv f_1 \cdots f_s$, and a set of correction factors $\alpha_1, \ldots, \alpha_s$ correct modulo p^b. We want to compute a factorization modulo p^{a+b} efficiently from this information.

We shall continue to assume that f and all the f_i are monic (actually it suffices to know what the leading coefficients will lift to). So the change to f_i is just $p^a\delta_i$ where $\delta_i := (\alpha_i(f - \prod f_j)/p^a) \bmod (p^b, f_i)$. The question is how to compute the δ_i quickly?

It is not too hard to spot that $p^a\delta_i \equiv (\alpha_i f) \bmod (p^{a+b}, f_i)$. So, in fact, we just reduce f modulo (p^{a+b}, f_i) directly, then multiply by α_i and reduce modulo (p^{a+b}, f_i) again — this gives δ_i. Note that each α_i is already reduced modulo (p^b, f_i).

Another scheme, used in REDUCE's factoriser, calculates the residue (i.e. $f(x) - \prod f_j(x) \in \mathbb{Z}[x]$) at the start and updates its value as the f_i are lifted; but this seems less efficient, especially as computing the residue in $\mathbb{Z}$ can generate needlessly large numbers: e.g. $x^2+1 \equiv (x+2057)(x+1068) \bmod 3125$ but the residue in $\mathbb{Z}[x]$ is $-3125x-2196875$ which has a coefficient greater than one million. In general, by considering the product of the constant terms we can see that if there are t extraneous modular factors then the residue in $\mathbb{Z}$ may have coefficients as large as p^{at}.

Lifting the Correction Factors

We showed earlier in this chapter that quadratic lifting is better than linear, provided that the correction factors could be lifted with about the same amount of work as the factors. We now study some ways of actually lifting the correction factors. Note that the correction factors are always lifted quadratically: the inputs to the algorithms are $f_1, \ldots, f_s$ modulo q^2 and $\alpha_1, \ldots, \alpha_s$ modulo q, and the result will be lifted correction factors $A_1, \ldots, A_s$ modulo q^2.

Several people have noticed the connection between this problem and that of partial fraction decomposition, namely:

$$\frac{1}{f_1 f_2 \cdots f_s} \equiv \frac{\alpha_1}{f_1} + \cdots + \frac{\alpha_s}{f_s} \bmod q.$$

This indicates that it might be worth looking at algorithms in that field, such as the one in [Kung&Tong77].

All the methods we consider lift the correction factors by finding the $\hat{f}_i$ and then using the lifting technique for reciprocals. We recall the remarkably simple way of lifting reciprocals: if $g \equiv f^{-1} \bmod q$ and we want to compute $h \equiv f^{-1} \bmod q^2$ then $h \equiv g(2-fg) \bmod q^2$ where we may take any representative of $g \bmod q^2$: we know $fg \equiv 1 \bmod q$ so we may define ε by $fg \equiv 1+q\varepsilon \bmod q^2$, now consider $hf \bmod q^2$ this is just $fg(2-fg) \equiv (1+q\varepsilon)(1-q\varepsilon) \equiv 1 \bmod q^2$. So provided we can find the $\hat{f}_i \bmod q^2$ efficiently, we can lift the α_i quickly.

A little more thought shows that we really need just $\hat{f}_i \bmod (q^2, f_i)$. We investigated six possibilities, some of which calculate $\hat{f}_i$ and others which produce $\hat{f}_i \bmod (q^2, f_i)$ directly. We label the methods (a) to (f).

(a) For each i divide f_i into $f \bmod q^2$ (should go exactly) then reduce the quotient modulo (q^2, f_i); or equivalently, reduce f modulo (q^2, f_i^2) then divide the remainder by f_i;

(b) For each i reduce all f_j $(j \neq i)$ modulo f_i then form their product modulo f_i;

(c) Calculate and store $f_1, f_1 f_2, \ldots, f_1 f_2 \cdots f_{s-1}$; also do the reverse list $f_s, f_s f_{s-1}, \ldots, f_s f_{s-1} \cdots f_2$. Form the products of one element from each list to generate the $\hat{f}_i$;

(d) split the f_i in a "balanced" fashion (try to balance either total degree or number of factors in each "half") and apply a divide-&-conquer approach like the algorithm in [Kung&Tong77].

(e) split the f_i in an "unbalanced" fashion, i.e. take the factor of least degree as one "half" and the rest as the other — this is a serial method: at the j^{th} iteration we compute $\prod_{i\neq 1}^{i\leq j} f_i, \prod_{i\neq 2}^{i\leq j} f_i, \ldots, \prod_{i\neq j}^{i\leq j} f_i$ and also $\prod_{i=1}^{i=j} f_i$;

(f) use the derivative of f as follows: we know $f \equiv \prod f_j \bmod q^2$ so $f' \equiv \sum f_j' \hat{f}_j \bmod q^2$, which implies that $f' \equiv f_j' \hat{f}_j \bmod (q^2, f_j)$; so $\hat{f}_j$ can be found easily — this was inspired by the Newton-Raphson iteration technique for root finding.

We expected (d), the balanced divide-and-conquer, to be best; however intuition can be misleading. We implemented all six methods of lifting and compared them on a variety of examples. To our surprise the seemingly inefficient method (b) of multiplying together lots of f_j turned out to be very good overall. A selection of the results is given in the table below:

Comparison of Correction Factor Lifting Methods						
Example	Time taken to perform the lifting					
[LLL82]	a	b	c	d	e	f
1	28.8	23.8	23.8	25.9	26.1	35.3
2	27.7	21.5	22.2	25.2	29.1	26.0
3	51.4	42.2	38.7	42.3	46.5	61.9
4	53.3	39.4	41.9	48.4	49.4	49.8
5	570	423	477	653	792	438
x^9-54	182	152	153	167	161	176

These times are interpreted PSL 3.4 on a Sun 3/50

A general complexity analysis of these algorithms does not produce a usable result, but if we restrict to the very special case when all the f_j have equal degree d we can get some interesting results. We assume it takes $(a+1)(b+1)$ basic multiplications to multiply two polynomials of degrees a and b; and similarly it takes $b(a-b+1)$ basic multiplications to form a quotient and/or remainder of a polynomial of degree a with a monic one of degree b. Armed with these two results we may proceed.

Method (a) is easy to analyse giving a total number of multiplications:

$$s(d(sd-d+1) + d((s-1)d-d+1)) = sd((2s-3)d+2).$$

Method (b) is almost as simple once we realise a multiplication modulo a polynomial is just a multiplication followed by remaindering. We find that (b) has complexity:

$$s(s\,d(d-d+1) + (s-2)[d^2+d(2(d-1)-d+1)]) = 2sd((s-2)d+1).$$

Observe that to reduce all the $\hat{f}_i$ to $\hat{f}_i$ mod f_i takes $sd((s-2)d+1)$ multiplications, so to beat method (b), any method which produces the $\hat{f}_i$ and then reduces these must find the $\hat{f}_i$ in less than $sd((s-2)d+1)$ multiplications.

Method (c) calculates $\hat{f}_i$ rather than $\hat{f}_i$ mod f_i; even so it takes longer than method (b):

$$\begin{aligned}&2((d+1)^2+(d+1)(2d+1)+\cdots+(d+1)((s-2)d+1)) +\\&(d+1)((s-2)d+1)+(2d+1)((s-3)d+1)+\cdots+((s-2)d+1)(d+1)\\&= (s-2)[(s-1)(s+6)d^2/6+(2s-1)d+2].\end{aligned}$$

To analyse method (d) we make a further simplifying assumption that s is a power of 2, say $s = 2^\sigma$. We define the function $\tau(\sigma)$ to be the number of multiplications taken by method (d) when there are 2^σ factors of degree d. We can write down a recurrence relation on $\tau(\sigma)$:

$$\tau(\sigma) = 2\tau(\sigma-1) + 2\times 2^{\sigma-1}\times(2^{\sigma-1}d+1)\times((2^{\sigma-1}-1)d+1) + (2^{\sigma-1}d+1)^2.$$

Converting $\tau(\sigma)$ into a summation we derive the closed form:

$$d^2 2^{\sigma-2}\left[\frac{2^{2\sigma+2}-1}{3}-2^{\sigma+1}+1\right] + d2^\sigma(2^{\sigma+1}-1) + 2^\sigma(\sigma+3)-1$$

Replacing each 2^σ by s gives the result

$$d^2\frac{s}{4}\left[\frac{4s^2-1}{3}-2s-1\right] + ds(2s-1) + s(3+\log_2 s)-1$$

which is again cubic in s.

Method (e) is a bit easier to analyse, but again it produces $\hat{f}_i$. At the j^{th} iteration the cost is

$$(j-1)[((j-2)d+1)(d+1) + (jd+1)(d+1)],$$

and summing this for $j = 1,2,\ldots,s-1$ yields the total number of multiplications:

$$\frac{2d^2(s-1)^2(s-3)}{3} + \frac{2s(s-1)(s-2)}{3} + \frac{(s-1)(s-2)}{2},$$

and once more this is cubic in s.

The differentiation method, (f), produces $\hat{f}_i f_i'$ mod f_i but is faster than (b): the number of multiplications being

$$sd + s(d((sd-1)-d+1) + d(2d-1)) = s(s+1)d^2.$$

However, it is less efficient because we must also lift the values of $(f_i')^{-1}$ mod f_i which nearly doubles the amount of work.

In summary we have shown that under the special circumstances of all factors having equal degree (and that there are 2^s of them for the divide and conquer method) that the multiplication method has best complexity, and our experimental results concur. In fact, the experimental results indicated that method (b) performs very well overall, consistently being the best or second best out of all six competitors.

7. Multivariate Hensel Lifting

The topic of this chapter is the deduction of the multivariate factorization of $f(x_1, \ldots, x_t)$ given the univariate factorization of $f(x_1, a_2, \ldots, a_t)$ where $a_2, \ldots, a_t$ are suitably chosen integers. We shall consider only fairly standard Hensel lifting techniques — in particular we ignore the recent lattice based methods, as these seem to be less practical currently and also because we have a generalisation only of the classical method for lifting factorisations over algebraic function fields.

We begin by looking at the standard algorithms of Musser and of Wang for factorization over $\mathbb{Z}$. We compare these methods and look at the practical problems that arise during the computations. These problems were discovered shortly after the first implementations were completed, and various strategies for alleviating or circumventing them have been published. We then look at possible extensions and adaptations of these methods to producing factorizations over algebraic number fields; in particular, we give an extension of Wang's leading coefficient prediction method [Wang78]. Extensions to algebraic function fields are discussed in the next chapter.

The Standard Algorithms

We surveyed several papers in this area in chapter 2, and here we shall be most interested in [Musser75], [Wang&Rothschild75], and [Wang78]. We present the algorithms proposed in these papers and then compare their merits paying attention to implementation issues. Finally we highlight the features used in our algorithm based on the foregoing comparison.

Musser's Algorithm

Input: a square-free primitive multivariate polynomial $f(x_1, \dots, x_t)$ over $\mathbb{Z}$;

Output: a list of the irreducible factors of f over $\mathbb{Z}$.

(1) Pick suitable substitution values $a_2, \dots, a_t \in \mathbb{Z}$ i.e. such that the image $f(x_1, a_2, \dots, a_t)$ remains square-free and retains full degree in x_1.

(2) Factorize the univariate polynomial $f(x_1, a_2, \dots, a_t)$ over $\mathbb{Z}$ to obtain factors $f_1, \dots, f_s$.

(3) Pick a large prime p (perhaps just smaller than the largest single-precision integer) such that $f(x_1, a_2, \dots, a_t)$ remains square-free and retains full degree in x_1; and reduce the f_i modulo p. So we have $f \equiv f_1 \cdots f_s$ mod $(p, x_2-a_2, \dots, x_t-a_t)$ — note that the f_i may be reducible modulo p, but they are all relatively prime.

(4) Let e_i be $1+\partial_{x_i}(f)$.

(5) For $j := 2, \dots, t$ do [lift the variables in succession]

(5.1) Quadratically lift the factorization to be valid modulo the ideal

$$(p, (x_2-a_2)^{e_2}, \dots, (x_j-a_j)^{e_j}, (x_{j+1}-a_{j+1}), \dots, (x_t-a_t)).$$

(6) Now lift quadratically in powers of p to obtain a factorization valid modulo $(p^{2^k}, (x_2-a_2)^{e_2}, \dots, (x_t-a_t)^{e_t})$ where p^{2^k} is larger than any coefficient which may occur in the factorisation.

(7) Finally determine the true factors via a combinatorial search — normally this should be trivial.

Wang & Rothschild's Algorithm

Input: a square-free primitive multivariate polynomial $f(x_1, \ldots, x_t)$ over $\mathbb{Z}$;

Output: a list of the irreducible factors of f over $\mathbb{Z}$.

(1) Pick suitable substitution values $a_2, \ldots, a_t \in \mathbb{Z}$ i.e. so that the homomorphic image $f \bmod (x_2-a_2, \ldots, x_t-a_t)$ has full degree in x_1 and remains square-free.

(2) Factorize the univariate polynomial $f(x_1, a_2, \ldots, a_t)$ over $\mathbb{Z}$ to obtain factors $f_1, \ldots, f_s$. So we have $f \equiv f_1 \cdots f_s \bmod J$ where J is the ideal $(x_2-a_2, \ldots, x_t-a_t)$ in the ring $\mathbb{Z}[x_2, \ldots, x_t]$.

(3) For $i := 2, \ldots, 1+$*total_degree*(f) do [lift homogeneous degree of the factors]

(3.1) Lift the factors f_j so that $f_1 \cdots f_s \equiv f \bmod J^i$

(4) Use a combinatorial search to find the true factors.

Wang's Improved (EEZ) Algorithm

This is almost identical to Musser's algorithm except that the prime chosen at step (3) is taken to be greater than a coefficient bound for the factors (thus making step(6) unnecessary), and that the lifting at step (5.1) is linear. The definition of "suitable" in step (1) has to be tightened: we want the homomorphic image to be of full degree and square-free and, in addition, we want the distinct irreducible factors of the coefficient of the highest power of x_1 each to have a "unique" prime divisor, that is for each of the distinct irreducible factors of this leading coefficient there is a prime which divides the homomorphic image of that factor alone — it has been emphasised [Norman&Moore81] that this extra condition may not be easy to fulfill. However, there is great benefit derived from Wang's coefficient prediction scheme and other heuristic tricks which can reduce the number of factors to be lifted — we discuss these below.

Comparison

The main difference is between Wang & Rothschild's algorithm and the other two; the distinction between the latter two being primarily a selection of devices to avoid extraneous factors or to detect factors early. We start by weighing the pros and cons of the two types of algorithm, then we talk about the various devices used in the EEZ algorithm.

Firstly we look at Wang & Rothschild's algorithm. It has a simpler structure than Musser's: the lifting is just in powers of a single ideal, and all the variables are lifted at once. The algorithm naturally works with polynomials as opposed to the apparent need for rational functions in Musser's approach. There are some disadvantages related to the behaviour on sparse polynomials: for example, reduction of a polynomial modulo the ideal J is hard unless all the a_i are zero (and rewriting in terms of $y_i := x_i+a_i$ will normally generate a dense polynomial), and also the lifting process itself generates dense intermediate results even when there are no extraneous factors (e.g. reducing $x^2y^2z^2+1$ modulo J^6, where J is the ideal $(x-1,y-2,z-3)$, gives us

$$6x^2y^2z-9x^2y^2+4x^2yz^2-24x^2yz+36x^2y-4x^2z^2+24x^2z-36x^2+2xy^2z^2-12xy^2z+18xy^2$$
$$-8xyz^2+48xyz-72xy+8xz^2-48xz+72x-y^2z^2+6y^2z-9y^2+4yz^2-24yz+36y-4z^2+24z-35$$

a completely dense polynomial of 26 terms with coefficients as large as 72). In general, reducing the polynomial $\prod_{j=1}^{t} x_j^n+1$ modulo J^{tn} produces a completely dense polynomial with n^t terms, where J is the ideal generated by $\{x_1-a_1, \ldots, x_t-a_t\}$.

As we have just commented, Musser's approach appears to need rational function arithmetic at first sight. Musser obviated this need by retaining ideal generators like $(x_2-a_2)^{e_2}$, so that the rational functions are represented as polynomials modulo these generators. He also introduced a numerical modulus p to avoid rational number arithmetic — we shall shortly see a reason for us not to do this. The variables are lifted one at a time, so we can take advantage of sparsity in the true factors: using the example above we get

$$\begin{aligned}
x^2y^2z^2+1 &\equiv 7 && \bmod ((x-1), (y-2), (z-3)) \\
&\equiv 12x-5 && \bmod ((x-1)^2, (y-2), (z-3)) \\
&\equiv 6x^2+1 && \bmod ((x-1)^3, (y-2), (z-3)) \\
&\equiv 36x^2y-36x^2+1 && \bmod ((x-1)^3, (y-2)^2, (z-3)) \\
&\equiv 9x^2y^2+1 && \bmod ((x-1)^3, (y-2)^3, (z-3)) \\
&\equiv 6x^2y^2z-9x^2y^2+1 && \bmod ((x-1)^3, (y-2)^3, (z-3)^2) \\
&\equiv x^2y^2z^2+1 && \bmod ((x-1)^3, (y-2)^3, (z-3)^3)
\end{aligned}$$

from which it is clear that we avoid the excessive intermediate expression growth inherent in Wang & Rothschild's method. We comment that even with this lifting scheme it is still possible for intermediate results to be denser than the factor we are lifting to, but only by a factor of $e-1$ where e is the degree of the factor in the variable that is currently being lifted. It should be noted that the sequence of ideals used as moduli in this lifting scheme is totally different from the sequence $J, J^2, J^3, \cdots$.

Musser uses quadratic lifting of the variables which is potentially hazardous because any extraneous factors are almost always dense — in other words lifting beyond the minimum necessary modulus could produce needlessly large intermediate results. For example, in the worst case we would lift to a factorization modulo $(p, (x_2-a_2)^{2e_2-2}, \ldots, (x_t-a_t)^{2e_t-2})$ which could give coefficients each having $2^{t-1}\prod(e_i-1)$ terms. Of course, if none of the factors is extraneous (as is normally the case) then such growth cannot occur if we lift too far.

Musser also suggests rewriting the polynomial to be factorised in terms of $y_i := x_i+a_i$ on the grounds that the ideal then has a very simple form allowing rapid calculation modulo the ideal. Again there is the risk that the substituted polynomial can have a number of terms exponential in the number of variables: for example, rewriting $x^2y^2z^2+1$ in terms of $X = x-1$, $Y = y-2$, and $Z = z-3$ gives a completely dense polynomial of 27 terms. Certainly, such a substitution would be infeasible if the input polynomial has high degree in many variables since the substituted polynomial could have as many as $\prod(1+\partial_{x_i} f)$ terms.

Wang's Tricks

Let us consider some of the tricks devised by Wang to expedite the lifting. Wang points out that by trying several sets of randomly chosen values for the a_i in step (2) [of Musser's algorithm] we can be almost certain that none of the factors is extraneous, thereby avoiding both the potentially expensive step (7) and the highly dense lifted factors — Wang suggests picking three sets of values as a good compromise between performing many univariate factorizations and getting the correct splitting pattern. He also observes that factors of low degree can be detected inexpensively at an early stage of the lifting process by performing trial divisions. This idea is better suited to linear lifting than to quadratic lifting because when quadratic lifting is used, each step performs more lifting than all the earlier steps together i.e. "early detection" of a factor of degree k, say, will only occur once all factors have been lifted to degree 2^s where s is the least integer such that $2^s > k$. Also the need to update the lifted correction factors when a factor is removed is dissuasive though the relevant calculations are quite simple: if f_i can be removed then each α_j for $j \neq i$ must be multiplied by f_i (and, of course, reduced modulo f_j); also we replace f by f / f_i. This explains Wang's choice of linear lifting at step (5.1).

Another significant contributor to the success of the EEZ algorithm is the predetermination of the leading coefficients by a clever trick: the leading coefficient is factorized, and then the values for the a_i are chosen so that for each of the distinct irreducible factors of the leading coefficient there is at least one prime dividing the image of that factor and none of the others. The factors of the leading coefficient can then be distributed correctly just by performing integer divisibility tests. When the factors are sufficiently sparse other coefficients can be deduced directly, possibly doing away with the need to lift at all.

The tricks just described could equally be applied to Wang & Rothschild's algorithm, but the "early detection" by trial divisions would not work out easily for

Musser's algorithm as the numerical modulus may be too small.

It should be pointed out that there is no use in lifting beyond half the maximum degree (either total degree [for Wang & Rothschild] or degree in the variable being lifted [for Musser]) since at most one of the factors has degree greater than half the maximum. In other words we can safely replace step (4) in Musser's algorithm by

(4) Let e_i be $1+\frac{1}{2}\partial_{x_i}(f)$.

Similarly in Wang & Rothschild's algorithm the loop starting at step (3) need only go as far as $1+\frac{1}{2}\mathit{total\_degree}(f)$.

Factorization over Algebraic Number Fields

The discussion above was made under the assumption that the factorization was happening over $\mathbb{Z}$. The situation is a little different when the coefficient domain is a ring of algebraic integers. Wang & Rothschild's algorithm works without modification (other than the change in factorization domain in step (2)). Musser's algorithm generalizes with no problem, except for the use of a numerical modulus p; the problem being that we cannot guarantee that the minimal polynomials of the extension generators remain irreducible modulo p. Our options include ignoring p, using some sophisticated conversion method from a finite field to an algebraic number field (e.g. Weinberger & Rothschild's, or Lenstra's), and using a heuristic method such as that in [Langemyr87]. The last approach is particularly well suited to this application: only those primes dividing the defect or a denominator in the canonical representation of the coefficients of the correction factors cause difficulty, any other primes may be used.

Regrettably, Wang's leading coefficient prediction trick generalises in a rather complicated way. The reason for this is that there may not be unique factorization in the ring of algebraic integers: for example in $\mathbb{Q}(\sqrt{-5})$ we have the distinct factorizations $6 = 2\times3 = (1+\sqrt{-5})\times(1-\sqrt{-5})$ yet all of 2, 3, $(1+\sqrt{-5})$ and $(1-\sqrt{-5})$ are irreducible. We do have unique factorization into ideals in the ring of integers but it is not clear how to apply this facility here.

Davenport [private communication] suggested taking norms and then factorizing the resulting integers. This neatly bypasses the non-unique factorization problem; however we cannot necessarily find integer values for the a_i such that the norms of the images of the distinct factors of the leading coefficient each have a prime factor not dividing any of the other norms: if the leading coefficient is $x_2^2 + x_3^2$ and the field is $\mathbb{Q}(i)$ then the leading coefficient factorizes into $(x+iy)(x-iy)$ and whatever integer values for a_2 and a_3 we pick, the two factors have equal norms.

However, by employing the argument in [Trager76] we can show that if we allow a_2 to be a general algebraic integer then we can force each norm to have its own prime factor as required by Wang's trick. What we do is compute the square-free decomposition of the leading coefficient and then pick integer values $a_3, \ldots, a_t$ so that the square-free components retain full degree in x_2 and remain square-free — let $g(x_2)$ denote the image of the leading coefficient. Form the product of the distinct irreducible factors of g as $h(x_2) := g(x_2)/gcd(g(x_2), g'(x_2))$. Applying theorem 2.3 in [Trager76] we can find an algebraic integer, β, such that $h(x_2-\beta)$ has a squarefree norm. We can then pick an integer value M for x_2 so that each of the factors of the norm of $h(x_2-\beta)$ has a prime factor not dividing any other factor. Thus using the value $M-\beta$ for a_2 completes a suitable set of substitution values for permitting use of Wang's prediction technique. In practice, a random algebraic integer value for a_2 should usually suffice.

If the above method is too complicated we can always resort to the simpler ways of eliminating the leading coefficient problem. We can force the polynomial being factorized to be monic by the obvious, though costly, substitution; or we can force all the factors to have leading coefficient equal to that of the original (effectively multiplying the original polynomial by a high power of its leading coefficient). These two ideas are usually deemed too inefficient on account of the potential growth encumbent in raising multivariate polynomials to high powers.

More recently Kaltofen has pointed out that the method described in [Kaltofen85a] for determining the leading coefficients applies (without modification) to our problem, and is probably the most efficient method. We give a short overview of Kaltofen's method. From the univariate factorization $f(x_1, a_2, \ldots, a_t) = f_1(x_1)f_2(x_1) \cdots f_s(x_1)$ lift to each possible bivariate factorisation: let the factorisation involving x_1 and x_k be $f_1^{(k)}f_2^{(k)} \cdots f_s^{(k)}$. We assume that there are no extraneous factors and that the leading coefficient of each of the bivariate factors is correct, i.e. the leading coefficient of $f_j^{(k)}$ is the image of the leading coefficient of f_j under the substitutions $x_r \rightarrow a_r \ \forall r \neq 1$ or k. Thus the list of leading coefficients from $f_1^{(k)}, \ldots, f_s^{(k)}$ is a factorization of the image of the leading coefficient of the original polynomial. Now we can recursively apply the method to each of these factorisations of the leading coefficient to obtain the true leading coefficients of $f_1, \ldots, f_s$, the factors of the original polynomial. Observe that some gcd computations may be needed to satisfy the coprimeness condition for Hensel lifting; and also observe that we may need to perform $t-1$ lifting operations from the univariate factorisation to yield all of the various bivariate factorizations because any factor of the leading coefficient involving just one variable, x_k say, can be correctly distributed from information contained solely in the factors $f_1^{(k)}, \ldots, f_s^{(k)}$ — such a factor would merely map to a field element in all the other homomorphic images.

Summary

We conclude by describing our algorithm for lifting from a univariate factorization to a multivariate one — selecting the best features of the methods we looked at. The algorithm is very similar to the EEZ method.

Our Algorithm

Input: a square-free primitive multivariate polynomial $f(x_1, \ldots, x_t)$ over $\mathbb{Z}$;

Output: a list of the irreducible factors of f over $\mathbb{Z}$.

(1) Pick suitable substitution values $a_2, \ldots, a_t \in \mathbb{Z}$ i.e. satisfying $f(x_1, a_2, \ldots, a_t)$ has full degree and is square-free.

(2) Factorize the univariate polynomial $f(x_1, a_2, \ldots, a_t)$ over $\mathbb{Z}$ to obtain factors $f_1, \ldots, f_s$. In fact, we repeat steps (1) and (2) for three (following Wang's recommendation) sets of substitution values as a ploy to avoid extraneous factors — see chapter 3.

(3) Determine the correct leading coefficients of the factors by Kaltofen's method: often we will need only a few of the bivariate lifts since, for example, if each factor of the leading coefficient of f involves at least one of x_2 or x_3 then the bivariate lifts for these two variables will be sufficient.

(4) Pick a suitable (large) prime p greater than twice the largest integer that may occur in any factor once denominators have been cleared — such a bound can be determined using Kronecker's substitution and a univariate coefficient bound for factors. An alternative is to use a heuristic bound, but this risks producing reducible factors should the heuristic fail. Reduce the f_i modulo p, so we have $f \equiv f_1 \cdots f_s \bmod (p, x_2-a_2, \ldots, x_t-a_t)$ — note that the f_i may be reducible modulo p, but they are all relatively prime.

(5) Let d_i be $\partial_{x_i}(f)$, and e_i be $\lceil \tfrac{1}{2} d_i \rceil$.

(6) For $j := 2, \ldots, t$ do [lift the variables in succession]

(6.1) Linearly lift the factorization to be valid modulo

$$(p, (x_2-a_2)^{d_2}, \ldots, (x_{j-1}-a_{j-1})^{d_{j-1}}, (x_j-a_j)^{e_j}, (x_{j+1}-a_{j+1}), \ldots, (x_t-a_t)).$$

Perform trial divisions on each lifting step once the exponent of (x_j-a_j)

exceeds d_j/s (average degree in x_j of the factors).

(6.2) Possibly one factor may have to be computed by division if it contains x_j to a power greater than e_j-1. After this final determination we have a factorisation correct modulo

$$(p, (x_2-a_2)^{d_2}, \ldots, (x_j-a_j)^{d_j}, (x_{j+1}-a_{j+1}), \ldots, (x_t-a_t)).$$

8. Factorization over Algebraic Function Fields

The aim of this chapter is to present and discuss a method for factorizing polynomials over algebraic function fields: for example, if $\alpha^2-4n-1=0$ then $x^2+x-n = (x+\frac{1}{2}(1+\alpha))(x+\frac{1}{2}(1-\alpha))$; and if $\alpha^2-(n+1)^3=0$ then $x^2-n-1 = (x+\alpha/(n+1))(x-\alpha/(n+1))$. The general paradigm will be to substitute suitable integer values for those transcendentals upon which the algebraic functions depend — thus converting the algebraic functions into algebraic numbers (possibly even rationals or integers). Then factorize the resulting polynomial over the algebraic number field corresponding to the original algebraic function field, and finally lift this factorization so that the true factorization can be found.

Notation

We set up some notation here for the rest of the chapter. The polynomial we wish to factorize is f. The field over which the factorization is to be made is $K := \mathbb{Q}(z_1, \ldots, z_\tau, \alpha_1, \ldots, \alpha_r)$ where the z_j are transcendentals and the α_j are algebraic. We assume the field has been built up via a succession of algebraic extensions giving us a tower starting from $K_0 := \mathbb{Q}(z_1, \ldots, z_\tau)$; for $j > 0$ $K_j := K_{j-1}(\alpha_j)$. Thus $K = K_r$. Let m_i be the minimal polynomial for α_i over the field K_{i-1} — this places an implicit ordering on the α_i as in the algebraic number case. We may also assume without loss of generality that each m_i is monic and has coefficients lying in the ring $\mathbb{Z}[z_1, \ldots, z_\tau, \alpha_1, \ldots, \alpha_{i-1}]$ — thus each α_j is an algebraic integer in a broader sense of the phrase (so, for example, the product of two polynomials over $\mathbb{Z}$ in the α_j will not contain any fractions).

There are two ways to view multivariate polynomials over algebraic function fields: as an essentially multivariate polynomial, or as a univariate polynomial (with all the variables except one absorbed into the algebraic function field). The latter view is the simpler; however, such a view is more restrictive in that we cannot in general find the content as an element of an algebraic function field: for example, over the field $\mathbb{Q}(z,\alpha)$ where $\alpha^2-z=0$ we might say that the polynomial $g(x) := (z^2-z)x+(z-1)(1-\alpha)$ has content $z-1$ because $g(x)=(z-1)(zx+1-\alpha)$, but equally we could say that it has content $z-\alpha$ because $g(x)=(z-\alpha)((z+\alpha)x+z-1)$. In contrast, the former viewpoint does allow us to find the content in terms of those variables not contained in the algebraic function field. This extra capability lead us to choose the former view.

Having chosen the multivariate representation, we now explain how we can reduce to the case of f being univariate over K. This is very simple: the lifting techniques of chapter 7 apply even when the coefficient domain is an algebraic function field; that is, we can choose integer substitution values for all except one of the variables to yield a univariate polynomial which we factorise, then we deduce the multivariate factorization.

Hence, for the remainder of this chapter we shall take $f(x)$ to be a univariate polynomial in $K[x]$. We also make the simplifying assumption that f is monic — if it is not, the best strategy appears to be to force all the factors to have leading coefficient equal to that of f; in this way we avoid the need to invert elements of K. The lack of unique factorization in K makes it essentially impossible to find the "true" leading coefficients of the factors (certainly Kaltofen's method cannot work as it requires the formation of a GCD-free basis).

Bounds on Coefficients of Factors

As for the earlier algorithms we need to find how large coefficients can get so we know how far to lift the modular factorization. The coefficients in an algebraic function field have two measures of size: the size of the numerical coefficients as well as the degree in the z_i.

Degree Bound

The first guess is a naive generalisation of the degree bound for factorization over number fields: the degree in each z_j of any factor is no larger than the degree in that same z_j of (any coefficient of) the input polynomial or of any coefficient in any minimal polynomial. Unfortunately there is a family of counter-examples to this guess: let p be an odd prime greater than 9, and let q be the largest integer less than $\sqrt{p}$ (so in particular, $q \geq 3$); now consider the algebraic function α defined by $(z+\alpha)^p = 2+z^q$, then the polynomial $x^p-(2+z^q)^q$ factorizes over $\mathbb{Q}(z, \alpha)$ as

$$(x-(z+\alpha)^q)(x^{p-1}+(z+\alpha)^q x^{p-2}+ \cdots +(z+\alpha)^{q(p-1)})$$

but $(z+\alpha)^{q(p-1)} = (z+\alpha)^{p(q-1)}(z+\alpha)^{p-q} = z^{p+q^2-2q}+ \cdots$ (lower powers of z). So we have a factor which has degree in z almost twice the maximum of the degrees in z of f and of the minimal polynomial of α — since $q \geq 3$ we have $q^2-2q > 0$.

Arguing along the lines of Trager's algorithm [Trager76] we can find upper bounds on the possible degrees in the z_j as follows. We define the degree in z_j of an algebraic function, α, with minimal polynomial $m_\alpha(y) = \sum_{i=0}^m b_i y^i$ to be

$$\partial_{z_j}\alpha := \max\left\{\frac{\partial_{z_j} b_i}{m-i}\right\}_{i=0}^{m}.$$

We extend ∂_{z_j} to K by defining the degree of a product to be the sum of the degrees of the terms, and the degree of a sum to be the maximum of the degrees of the summands. For example, if α has minimal polynomial $m_\alpha(y) = y^2-z$ and β has minimal polynomial $m_\beta(y) = y^2-\alpha z$ then $\partial_z\alpha = 1/2$ and $\partial_z\beta = 3/4$. This definition of degree is just a simple upper bound on the order of the pole (or the zero, if negative) in α as z_j tends to infinity. The following example shows that ∂_{z_j} can be different from the order at infinity. Let $\alpha^2-z-1=0$, and $\beta^2-z-2=0$. So $\partial_z(\alpha)=\partial_z(\beta)=1/2$. Then according to the definition $\partial_z(\alpha-\beta)=1/2$, yet our conventions lead us to think that $\alpha-\beta$ has a zero of order 1/2 at infinity (i.e. a pole of order −1/2). This definition of degree takes the maximum over all possible choices of roots — both α and $-\alpha$ are square-roots of $z+1$. Note that with the restrictions placed on the minimal polynomials of the algebraic functions our degree function is always non-negative on algebraic kernels.

Now we can work through Trager's method to find our bound: we may have to make a linear substitution ($x \to x+\delta$) in $f(x) := \sum_{i=0}^{n} a_i x^i$ to obtain a polynomial which has a square-free norm. The degree in each z_j of the linear shift, δ, is at most the maximum of the degrees in z_j of the algebraic functions $\alpha_1, \ldots, \alpha_r$ since δ is only a $\mathbb{Z}$-linear combination of the algebraic generators. So we find that an upper bound for the degree of any coefficient of the shifted polynomial is

$$\partial_{z_j}(f(x+\delta)) \le \max_{i=0,\ldots,n} \left\{ i\partial_{z_j}\delta + \partial_{z_j}a_i \right\}$$

Finally, this quantity must be multiplied by the extension degree to produce a bound on the degree of the square-free norm. Unfortunately this is often far too large: in the counter-example above the bound turns out to be p^2 which is much larger than $2p > p+q^2-2q$.

A good heuristic bound for the degree in z_j of the factors seems to be $\partial_{z_j}f + \sum_i \partial_{z_j} m_i$. This gives the more reasonable figure of $p+q^2$ in the counter-example above. We should explain that we have no justification for the validity of this bound in the general case; however, we are unaware of any situations where it is invalid.

Wang's idea of using early detection is still to be recommended because even the heuristic bound may be far too large.

Numerical Bounds

Again we can follow through Trager's algorithm estimating the largest possible integers that may occur: let n be the degree of f, and d_i be the degree of m_i (i.e. d_i is the extension degree $[K_i:K_{i-1}]$), then from the proof of theorem 2.3 in [Trager76] we find that the shift $\delta =: s_1\alpha_1 + \cdots + s_r\alpha_r$ can be constrained to have $|s_k| \le ½d_k^2n^2$. This gives sufficient information for us to compute an upper bound on the sizes of numerical coefficients in the square-free norm from which we can derive upper bounds on the sizes of coefficients of the factors of the norm. Finally we must compute a bound on the sizes of the coefficients in the factors of the original polynomial, and this we do by estimating coefficient sizes during a polynomial remainder sequence. Note that the factors of the norm must have degree divisible by the extension degree $[K_r:K_0]$ (since the factors of the norm are themselves norms), and this may help lower the bound. Unfortunately, this method gives bounds that are far too large: for example, if $f(x) = x^2+x-z$ and $\alpha^2-4z-1=0$ then we know the true factors are $x+½(1+\alpha)$ and $x+½(1-\alpha)$ but the bounding process gives $|s_1| \le 8$, so the square-free norm has coefficients with magnitude at most 256, and Gel'fond's bound (e.g. [Wang75]) for the factors of the norm is almost 281000; lastly we simulate a polynomial remainder sequence estimating degree and numerical bounds to get the bound for the factors of f to be greater than 5000000.

Strictly we have no need for a numerical bound as the lifting can be performed without numerical modulus, but as observed in [Musser75] this leads to calculations with rational numbers, thereby incurring the cost of integer gcd computations. In spite of this, it seems best not to use a numerical modulus until there is a significant improvement in the tightness of the coefficient bound. We can decide whether to use rationals or modular numbers based on the tightness of our bound: suppose computing the gcd of two n-digit integers takes as much time as κ multiplications of n-digit integers then if our multiplication algorithm is quadratic then it is better to use modular arithmetic only if our bound has length at most about $\sqrt{\kappa+2}$ times the size of the numbers appearing in the rational number computation.

Substitution Values

We consider two attitudes to selecting the integers, a_i, we shall be substituting for the transcendentals z_i. One is to have fairly weak restrictions on the permitted values but then have to work hard during the reconstruction of the true factors from the modular ones. The other attitude is to place stringent conditions on the permitted values which lead to a less onerous reconstruction.

Stringent Case

Let us first look at the case where we are more restrictive in our choice of substitution values. Ideally, we want values which would lead to a trivial reconstruction.

We begin by making some definitions. Let $a_1, \ldots, a_\tau \in \mathbb{Z}$ be the chosen substitution values for $z_1, \ldots, z_\tau$ respectively. We observe that the algebraic functions become algebraic integers under this substitution: the minimal polynomial for α_1 maps to a monic polynomial with integer coefficients, and the minimal polynomials for the other α_i inductively are monic and have algebraic integer coefficients. Let β_i be the algebraic integer to which α_i maps. We can show this situation neatly in a diagram:

$$\begin{tikzcd}
K_r = K_{r-1}(\alpha_{r-1}) \arrow[r] & M_r = M_{r-1}(\beta_{r-1}) \\
\vdots \arrow[u] \arrow[r, phantom, "\vdots"] & \vdots \arrow[u] \\
K_1 = K_0(\alpha_1) \arrow[u] \arrow[r] & M_1 = M_0(\beta_1) \arrow[u] \\
K_0 = \mathbb{Q}(z_1, \ldots, z_\tau) \arrow[u] \arrow[r] & M_0 = \mathbb{Q} \arrow[u]
\end{tikzcd}$$

where the K_i are the elements of the original tower, and the M_i are their images under the substitution $z_j \to a_j$.

A good starting point is to consider the images, β_j, of the algebraic functions α_j. To make the reconstruction trivial it would be advantageous if each algebraic integer β_j had the same degree as its pre-image α_j — i.e. we want the image of the minimal polynomial of each α_j to be irreducible over M_{j-1}, so that β_j is defined as a root of the image of $m_j(y)$ in $M_j[y]$. Note that with this restriction the M_i are uniquely determined by the a_j. Also, we want to be certain that the substitution produces no extraneous factors.

Luckily, we can meet all these conditions: hardly surprising since we have a lot of freedom to choose the a_i. In fact, by picking random values for the a_i from a sufficiently wide range we can be almost certain that the conditions will be satisfied. To show this we employ a primitive element for K — the theorem of the primitive element guarantees its existence, and [Trager76] presents an algorithm for computing one (we do not actually need to compute a primitive element but we shall refer to the algorithm). Let α be any primitive element for K over K_0, and let m_α be its minimal polynomial over K_0. So $K = K_0(\alpha)$ and we may apply the factorisation algorithm in [Trager76] which proceeds by computing a square-free norm and then showing that this has a factorisation over K_0 which corresponds directly to the factorisation of the original polynomial over the original field. Our argument uses this fact to show that there is a choice of substitution values with the required properties.

We can apply Trager's square-free norm algorithm to produce a polynomial in $g(x) \in \mathbb{Q}[z_1, \dots, z_t][x]$ whose factorization over $\mathbb{Q}$ corresponds to the true factorization of f. An effective version of Hilbert's irreducibility theorem presented in [Kaltofen84] enables us to reduce to a bivariate polynomial easily: it says that a random vector $(a_3, \dots, a_t) \in \mathbb{Z}^{t-2}$ with $|a_i| < M$ will (with controllably high probability, $1-4\delta 2^\delta/M^{t-1}$ where δ is the total degree of f) keep the factors of g irreducible over $\mathbb{Q}$ under the homomorphism $\forall j \geq 3\ z_j \to a_j$. Then the last step to a univariate polynomial can be achieved by, say, a result in [Fried74]; i.e. we can almost certainly ensure the complete absence of extraneous factors. Any such set of substitution values will automatically keep m_α irreducible because otherwise extraneous factors would appear — recall that norms can be calculated using resultants and that $res(fg, h) = res(f, h)res(g, h)$, so if one of the minimal polynomials mapped to a reducible polynomial then all the irreducible factors of the square-free norm would map to reducible factors.

We assume that at least one suitable set of substitution values can be found — picking random values for $a_3, \ldots, a_t$ (the range 0 to $2^{24}-1$ as supplied in Cambridge LISP is fine for all practical purposes) and then computing a suitable value for a_2 will probably find a valid set straightaway, if not we just pick another random set. Indeed, Kaltofen privately suggested we use random values (e.g. between 0 and $2^{24}-1$) for the a_i and then assume that all the conditions were met without bothering to check them, and thus saving a great deal of effort. In the unlikely event that one of the conditions be violated (e.g. minimal polynomials of some of the algebraic functions mapping to reducible polynomials) we will discover it during the factorization process and only then pick a new random set.

The rigorous alternative is to check for each i that the image of the minimal polynomial m_i remains irreducible over M_{i-1}, and that the image of f is square-free. [Musser78] suggests that we can check irreducibility with short expected time just by factorising in a few finite fields and performing a degree compatibility check, with the possibility of having to perform a complete factorization in unlucky cases [KMS83]. Of course, checking f for square-freeness is not hard.

Once we have a valid set of a_i and a factorization of the homomorphic image of f, we have only to lift the factors. The factors are assumed all to lift to true factors so we need not worry about extraneous factors (nor leading coefficients as these are all forced equal to that of f). However, we *do* still have to worry about denominators appearing in the coefficients, as in the example given near the start of this chapter. The arguments in chapter 4 giving a couple of denominator bounds apply equally to algebraic function fields: the square of the common denominator appearing in the representation of any algebraic integer with respect to a particular basis divides the discriminant of that basis (the discriminant formula is valid for algebraic function fields too); alternatively, we can compute an integral basis to get an exact result (this involves even lengthier computations than for algebraic number fields). Either way, we get an integer and a polynomial in $\mathbb{Z}[z_1, \ldots, z_\tau]$ which can be used to clear all denominators.

The Hensel lifting proceeds much as for normal multivariate lifting (chapter 7) except that the minimal polynomials of each β_j must be lifted along with the factors. Once the Hensel lifting has been completed, we just replace each β_j by α_j to obtain the factorization of f over K. We illustrate this with a short example:

to factorise $f(x) := x^2-z^4+2z^2\alpha-z$ over $K := \mathbb{Q}(\alpha)$ where $m_\alpha(\alpha) := \alpha^2-z = 0$; we might pick the substitution value 2 for z (any non-square integer will do);
α maps to β with minimal polynomial $m_\beta(x) := x^2-2 = 0$, thus
K maps to $\mathbb{Q}(\beta)$, and
f maps to $g(x) = x^2-18+8\beta$.
g factorizes into $(x+4-\beta)(x-4+\beta)$ over $\mathbb{Q}(\beta)$;
we regard this as a factorization modulo $(z-2)$, and lift the factorization in powers of this ideal:

$f(x) \equiv (x+4z-4-\beta)(x-4z+4+\beta) \bmod (z-2)^2$,

$m_\alpha(\beta) \equiv \beta^2-z \bmod (z-2)^2$;

$f(x) \equiv (x+z^2-\beta)(x-z^2+\beta) \bmod (z-2)^3$,

$m_\alpha(\beta) \equiv \beta^2-z \bmod (z-2)^3$.

Hence $f(x) = (x+z^2-\alpha)(x-z^2+\alpha)$ is the factorization of f into irreducibles over K.

In summary, we can probably reduce the algebraic function field factorization problem to an algebraic number field one with very little effort. The algebraic number field factorization has to be over a field of the same extension degree as the original algebraic function field was over K_0; the coefficients of the image polynomial tend to be large because the images of the z_i have to be chosen from such a wide range. Like the reduction, the lifting process and reconstruction of the factors is quite simple. Clearly, the running time is dominated by the factorization for all but the smallest extensions. Now we turn to another method.

Lax Case

Here we investigate the possibility of allowing choices for the a_i which do not keep the minimal polynomials, m_i, irreducible. We still insist that f and all the m_i remain square-free so that we can apply Hensel lifting later on — recall that f and the m_i are all assumed to be monic, so no extra conditions about non-vanishing leading coefficients are needed. The purpose behind looking at this more complicated variant is to avoid the need to factorize over a field of such high extension degree; instead the lifting stage has to work harder to make up for the "lost" algebraic extensions.

When any image (modulo the evaluation ideal) of a minimal polynomial, m_j, becomes reducible we can apply generalizations of the methods used with algebraic number fields where a generator has a minimal polynomial which factorises modulo the chosen prime. There we had two possible approaches: one was Weinberger & Rothschild's which used the Chinese Remainder Theorem to recover the true answer, the other was Lenstra's which deduced the true answer from a factorization derived by using just one of the factors of the minimal polynomial to generate the finite field. The disadvantages of possibly super-exponential behaviour (see chapter 3) inherent in Weinberger & Rothschild's approach still remain for this application, so we adopt a method akin to Lenstra's.

We pick one of the irreducible factors, $\overline{m}_j$, of the image of m_j to use as the minimal polynomial of β_j, the algebraic integer which will correspond to α_j. The new tower of extensions can be represented with a similar diagram to that used for the stringent case:

$$\begin{tikzcd}
K_r = K_{r-1}(\alpha_{r-1}) \arrow[r] & M_r = M_{r-1}(\beta_{r-1}) \\
\vdots \arrow[u] & \vdots \arrow[u, "\le d_r \text{ extensions}"] \\
K_1 = K_0(\alpha_1) \arrow[u] \arrow[r] & M_1 = M_0(\beta_1) \arrow[u, "\le d_2 \text{ extensions}"] \\
K_0 = \mathbb{Q}(z_1, \dots, z_\tau) \arrow[u] \arrow[r] & M_0 = \mathbb{Q} \arrow[u, "\le d_1 \text{ extensions}"]
\end{tikzcd}$$

Note that in this situation the M_i are not necessarily uniquely determined, so we make some consistent set of choices.

Using methods from our stringent case, we can factorize the homomorphic image of f over $M_r = \mathbb{Q}(\beta_1, \dots, \beta_r)$, and then lift the factors to be correct modulo $I := ((z_2-a_2)^{e_2}, \dots, (z_\tau-a_\tau)^{e_\tau})$. To be able to deduce the coefficients of the true factors we must lift far enough that

$$S := \{z_2^{j_2} \cdots z_\tau^{j_\tau} \alpha_1^{k_1} \cdots \alpha_r^{k_r} : 0 \le j_s \le B_s \text{ and } 0 \le k_s < \partial\alpha_s\}$$

maps to a $\mathbb{Q}$-linearly independent set modulo I, where B_s is a degree bound for z_s in any coefficient of any factor. Clearly $e_i \ge \max\{B_i, \partial_{z_i} m_j\}$ but we have not yet been able to determine a sufficient value. Observe that S is a $\mathbb{Q}$-basis for our field K.

If a sufficient value for each e_i can be found then the coefficients of the true factors can be found by solving the system of linear equations derived from the images of the elements of S. We give the same example as we did for the stringent case to show how the method works.

To factorize $f(x) := x^2-z^4+2z^2\alpha-z$ over $K := \mathbb{Q}(\alpha)$ where $m_\alpha(\alpha) := \alpha^2-z = 0$; we might pick the substitution value 4 for z (we allow square integers now); $m_\alpha(x)$ maps to x^2-4 which is reducible over $\mathbb{Q}$: $x^2-4 = (x-2)(x+2)$. We must choose one of these factors as the minimal polynomial for β (the image of α). Let us pick $m_\beta(x) = x-2$ — in effect α maps to 2. So f maps to $g(x) := x^2-196$ and K maps to $\mathbb{Q}$. We can factorize the image of f over the image of K: $g(x) = (x-14)(x+14)$. In this case there are no extraneous factors of g, and we have already commented that a random choice of substitution values almost certainly avoids them in general.

We find how far we must lift the factorization by estimating the maximum degree in z of any coefficient of any factor: in this example we use the crystal ball algorithm to estimate the degree bound at 2. So each coefficient will be a $\mathbb{Q}$-linear combination of 1, z, z^2, α, αz, αz^2. Hence we must lift far enough to get at least 6 degrees of freedom (where degrees of freedom is extension degree of algebraic number field times the product of the powers of the ideal generators). In this case we lift to $(z-4)^6$ and can easily verify that the $\mathbb{Q}$-basis for coefficients remains linearly independent modulo $(z-4)^6$. We also find that

$$\beta \equiv \frac{7z^5-180z^4+2016z^3-13440z^2+80640z+64512}{131072} \bmod (z-4)^6$$

and

$$g(x) \equiv \left[x+\frac{7z^5-180z^4+2016z^3-144512z^2+80640z+64512}{131072}\right]$$
$$\left[x-\frac{7z^5-180z^4+2016z^3-144512z^2+80640z+64512}{131072}\right] \bmod (z-4)^6.$$

From this we immediately spot the true factorisation: $f(x) = (x-z^2+\alpha)(x+z^2-\alpha)$, though in general it would have to be deduced by inverting the linear map.

It can be seen from this example that we needed to factorize only over $\mathbb{Q}$ even though the ultimate factorization was over an algebraic field. We can also see there are some definite disadvantages to this method. In particular, whenever the image of m_j is reducible $\overline{m}_j$ is necessarily an extraneous factor and so will lead to increasingly dense intermediate results (the lifts of $\overline{m}_j$ modulo high powers of the ideal) if standard Hensel lifting is used. This behaviour was becoming apparent even in the small example above:

just look at the image of α modulo $(z-4)^6$! Also if there are many z_j then then the linear system to be inverted will be exceedingly large — this fact alone precludes application of a "lax" method to moderately large problems.

Conclusion

We have presented two ways of achieving what we set out to do. One is a probabilistic method with the drawback of requiring the factorization of a polynomial with large coefficients over an algebraic number field with extension degree equal to that of the algebraic function field. The other lacks a proof of how far the Hensel lifting must go, but requires a factorization over an algebraic number field of much lower extension degree than the algebraic function field. Also the latter allows smaller substitution values which will help expedite the factorization. Neither algorithm seems at first sight to be superior to the other; the second method will need more space since it necessarily suffers from intermediate expression growth during the lifting stage.

The two methods considered in this chapter are of little practical value as they stand because both of them are too slow when solving anything but the very smallest of factorization problems. Thus there is plenty of scope here for further work, and development of efficient algorithms: for example, determination of tighter bounds on the coefficients would be very worthwhile. A fast deterministic method of picking small substitution values suitable for the first method would give a truly viable algorithm for factorization over algebraic function fields. Even better would be an algorithm that could reduce the algebraic function field factorization to a relatively easy factorization over an algebraic number field of small extension degree over $\mathbb{Q}$, and then retrieve the factors of the original polynomial possibly using a modified lattice reduction algorithm.

9. Conclusion

We now reflect on what has been discussed in chapters 2 to 8; we summarize our findings, and suggest areas for further study. The broadest conclusion is that, using a "classical" approach like ours, one can implement a suite of routines for the factorisation of a polynomial over an algebraic number field. The package will be fast enough to factorize polynomials of degree up to about 20 over algebraic number fields of extension degree (over $\mathbb{Q}$) up to about 10 in a reasonable time on a machine with the power of a Sun 3/160, say. The results in appendix F demonstrate our algorithm is definitely superior to Trager's [Trager76]; and according to [Lenstra82], Weinberger & Rothschild's method is inferior to the algorithm from which we developed ours—also the MACSYMA group at MIT found Trager's method to be superior to Weinberger and Rothschild's.

The extensions to factorization over algebraic function fields are still too slow to be useful. A sparse lifting method is required to achieve the conversion from a factorization over an algebraic number field to one over an algebraic function field.

These are succinct statements of what we have discovered:

- It is worth computing the optimal denominator bound (via integral bases).
- We have tightened the numerator bound, but there is room for further improvement.
- The "block" version of Lovász's algorithm is consistently quick. We have subsequently extended the rational number recovery method [WGD82] to algebraic number fields; this uses Lovász's algorithm.

- Choice of prime: the finite field factorization and lattice basis algorithms have opposing requirements. The decision should be determined by estimating the total factorization time for each choice.
- Univariate factor lifting is best achieved by our own truncated quadratic method.
- Trial divisions during the recombination of univariate factors should use an "early abort" scheme
- Multivariate lifting should be performed with a sparse algorithm, but this does not generalise easily to determination of factorizations over algebraic function fields.

The next few paragraphs expand on these succinct statements.

In chapter 4, we observed the importance of determining good coefficient bounds, and saw to our surprise that it is worth expending the effort of finding the optimal denominator bound by computing an integral basis. The effect of using a tight numerator bound (using foresight) can be even more dramatic (see the table on page 4.13), but the bound we currently use is far too large. So a method of obtaining a tighter numerator bound will enable factorizations to be found much faster because of the consequent reduction in both Hensel lifting time and basis reduction time. A possible avenue is to use approximations to the images in $\mathbb{C}$ of the algebraic numbers involved, though this can be ill-conditioned. Recently a promising new result has appeared in [CMP87].

The importance of good bounds is re-emphasised in chapter 8 where we the bounds for factorisation over algebraic function fields are especially poor. There we were forced to use trial divisions (over an algebraic function field) during the lifting process. We also showed that to employ a numerical modulus to our advantage during the lifting process needs quite a tight upper bound on the numerical size of possible factors—we know of no suitable bound.

The next centre of attention was the lattice basis reduction algorithm: the keystone of the entire algorithm. We saw in chapter 5 that for factorizations over large (degree > 3) algebraic number fields most of the time is spent in this algorithm. We also developed some improvements to the reduction algorithm making it significantly faster: indeed, the clear-cut superiority over Trager's algorithm depends on these improvements. Even so, whenever the minimal polynomial(s) become reducible mod p the basis reduction is still a highly time-consuming step during a factorization. Thus further improvements to the basis reduction algorithm would certainly have a sizeable impact on the overall efficiency. Consequently, there ought to be an investigation into how to adapt the basis reduction algorithm so that it takes advantage of the special structure of the bases that occur in our applications.

Subsequently to the work for this thesis we have found an extension of the rational number reconstruction algorithm (e.g. [WGD82]) to algebraic number fields. This permits use of an upper bound on the denominator which is not a multiple of the true denominator: for example, the square-root of the discriminant. The algorithm is very similar to Lenstra's reconstruction; this suggests it would still be quicker to compute an integral basis.

We can explain qualitatively why a finite field favourable for the lattice reduction is unfavourable for the finite factorization, and vice versa. It is easier to compute factorizations over smaller finite fields but the reconstruction (i.e. lattice reduction) always has to "lift" to the full extension degree — this manifests itself as smaller finite fields giving rise to initial bases with larger orthogonality defect. So the choice of finite field is a compromise, which is best determined by computing an estimate of the total factorization time for a few candidate fields and then picking the most promising of those. Unfortunately, this will be depend on the details of the finite field factorization, basis reduction and Hensel lifting routines. However, we have an empirical complexity formula for the basis reduction routine.

The remaining sections of the factorizer which consume a significant amount of processing time are the lifting stages. Our algorithms for lifting the *univariate* factorization appear to be as fast as possible, but the algorithm for lifting from a univariate factorization to a multivariate one is based on an "old" method. The principal disadvantage of this general scheme is that it suffers from "fill-in" (i.e. it produces dense intermediate results even though the input and final output may both be sparse): in chapter 8 we saw that the method can generate excessively large intermediate results even when there are no extraneous factors, and worse still, if the "lax" viewpoint was taken (when the minimal polynomials map to reducible images; page 8.10), then this pathological behaviour is guaranteed to occur!

Certainly the more modern sparse lifting methods can readily be adapted to operate over algebraic number fields; however, the necessary modifications to allow them to lift from a factorization over an algebraic number field to the corresponding factorization over an algebraic function field are far from obvious. Development of the necessary modifications seems a fruitful area for continued research, and would certainly lead to a viable factorization algorithm for algebraic function fields.

Theoretically the most time-consuming stage of our factorization process is the part where combinations of modular factors are multiplied together and then converted to putative true factors which are then verified or discarded. Under the worst conditions this has exponential complexity, though under normal circumstances only a small proportion of the total time is spent trying combinations. Our intention was to produce a useful tool for finding factorizations over algebraic fields rather than to construct an asymptotically fast algorithm. By applying sparse lifting techniques to the problem of chapter 8, one may be able to build a polynomial time algorithm based on a published polynomial time univariate factorization algorithm. This looks like an area worthy of further research.

Before summarising our algorithm we return to one of our original motivations: symbolic integration. We find that our algorithm has somewhat limited applicability because even quite small integration problems can lead to huge factorization problems: e.g. to integrate a univariate rational function with quintic denominator is infeasible since in general it requires the factorization of a quadratic over an extension field of degree 60. This enormous growth is inherent in the integration problem because a splitting field will normally have extension degree equal to the factorial of the degree of the polynomial we wish to split (i.e. the extension degree can be super-exponential in the degree of the polynomial).

Recent papers have proposed ways of solving larger integration problems by permitting the solution to contain expressions of the form:

$$\sum_{f(\alpha)=0} g(\alpha)\log h(x,\alpha)$$

where the factorization over a splitting field is implicit. Of course, we can obtain an explicit answer only by factorizing over the splitting field.

One area that stands to gain considerably from the development of an efficient factorizer is that of computing a Gröbner basis. Davenport has reported [Davenport87] some extremely encouraging results from performing factorizations during the determination of a Gröbner basis. Gröbner bases have lately become highly important in computer algebra where many problems (e.g. ideal membership, robot kinematics and geometrical theorem proving) can be solved quite simply given an algorithm to calculate a Gröbner basis.

Our Recommended Algorithm

We give just the algorithm for factorization over algebraic number fields.

Multivariate Factorization

Input: a multivariate polynomial $f(x_1, \ldots, x_n) \in K[x_1, \ldots, x_n]$ where K is an algebraic number field.

Output: the irreducible factors of f over K.

(1) perform content and square-free decompositions on f, then apply the following steps to each component.

(2) map down to a univariate factorization problem by picking suitable substitution values $a_2, \ldots, a_n$ for $x_2, \ldots, x_n$.

(3) compute the univariate factorization using the algorithm below.

(4) determine leading coefficients using Kaltofen's method (see page 7.9), and use a sparse lifting method to complete the factorization (e.g. [Zippel79] or [Zippel81]).

Univariate Factorization

Input: a univariate polynomial $f(x) \in K[x]$ where K is an algebraic number field

Output: the irreducible actors of f over K.

(1) Form the monic square-free components of f; do the following to each component.

(2) Compute an integral basis for K and thus obtain the optimal denominator bound (see page 4.5).

(3) Compute a numerator bound: bound the magnitude of roots of f in $\mathbb{C}$ using rb (page 4.9); use the binomial theorem (page 4.11) to get an upper bound on the magnitude of the coefficients of any factor; take the smaller of the Hadamard and Landau-Mignotte bounds (page 4.12).

(4) Try a few primes (not diving the denominator bound) and pick the one giving smallest factorization time estimate. Apply the Cantor-Zassenhaus algorithm to find the factors mod p.

(5) Use truncated quadratic lifting (page 6.5) to get the factors mod p^k for k sufficiently large. Details of lifting algorithms are on pages 6.9 & 6.10

(6) Use the "block" variant (page 5.11) of Lovász's algorithm to compute the LLL-reduced basis preparatory to conversion (page 5.4) of the modular factors.

(7) Recombine the factors in cardinality order (pages 3.8 & 3.9), using early abort trial division (page 3.9)—we can also test the putative factors for sufficiently small coefficients by modifying r and j on page 4.11

10. References

[Abbott&Davenport88] J A Abbott and J H Davenport, "A Note on a Paper by Wang: Another Surprising Property of 42," *Math Comp* **51**(184), pp 837-839

[ABD85] J A Abbott, R J Bradford and J H Davenport, "A Remark on Factorization," *SIGSAM Bulletin* **19** (May 1985) pp 31-33 & 37

[ABD86] J A Abbott, R J Bradford and J H Davenport, "The Bath Algebraic Number Package," *Proc SYMSAC 86* (Waterloo) pp 250-253

[ACP77] S K Abdali, B F Caviness and A Pridor, "Modular Polynomial Arithmetic in Partial Fraction Decomposition," *Proc 1977 MACSYMA User's Conf* pp 253-261

[A&G84] L Afflerbach and H Grothe, "Calculation of Minkowski-Reduced Lattice Bases," *Computing* **35** pp 269-276

[Berlekamp67] E R Berlekamp, "Factoring Polynomial Over Finite Fields," *Bell System Technical Journal* **46** pp 1853-1859

[Berlekamp70] E R Berlekamp, "Factoring Polynomials over Large Finite Fields," *Math Comp* **24**(111) pp 713-735

[Bradford88] R J Bradford, "On the Computation Of Integral Bases and Defects of Integrity," PhD thesis, Univ of Bath

[Calmet&Loos82] J Calmet and R Loos, "Deterministic versus Probabilistic Factorization of Integral Polynomials," *Proc EUROCAM 82* (Marseille) Springer LNCS 144 pp 117-125

[CZ81] D G Cantor and H Zassenhaus, "A New Algorithm for Factoring Polynomials over Finite Fields," *Math Comp* **36**(154) pp 587-592

[CMP87] L Cerlienco, M Mignotte, and F Piras, "Computing the Measure of a Polynomial," *JSC* **4** pp 21-33

[Collins79] G E Collins, "Factoring Univariate Integral Polynomials in Polynomial Average Time," *Proc EUROSAM 79* (Marseille) Springer LNCS 72 pp 317-329

[Coppersmith&Davenport85] D Coppersmith and J H Davenport, "An Application of Factoring," *J Symb Comp* **1** pp 241-243

[Davenport87] J H Davenport, "Looking at a Set of Equations," Bath Computer Science Technical Report 87-06

[D&T81] J H Davenport and B M Trager, "Factorization over Finitely Generated Fields," *Proc SYMSAC 81* (Snowbird) pp 200-205

[Dieter75] U Dieter, "How to Calculate Shortest Vectors in a Lattice," *Math Comp* **29**(131) pp 827-833

[F&P85] U Fincke and M Pohst, "Improved Methods for Calculating Vectors of Short Length in a Lattice, Including a Complexity Analysis," *Math Comp* **44**(170) pp 463-471

[Fried74] M Fried, "On Hilbert's Irreducibility Theorem," *JNT* **6** (1974) pp 211-231

[G&T85] P Gianni and B Trager, "Gcds and Factoring Multivariate Polynomials Using Gröbner Bases," *Proc EUROCAL 85* (Linz) Springer LNCS 204 pp 409-410

[vzG&K85a] J von zur Gathen and E Kaltofen, "Factorization of Multivariate Polynomials over Finite Fields," *Math Comp* **45** (Jul 1985) pp 251-261

[vzG&K85b] J von zur Gathen and E Kaltofen, "Factoring Sparse Multivariate Polynomials," *J Comp & Sys Sci* **31** (Oct 1985)

[Goebbels85] F Goebbels, "Factorization of Rational Polynomials in the Zassenhaus Norm," *Proc EUROCAL 85* (Linz) Springer LNCS 204 pp 146-147

[G&A81] H Gunji and D Arnon, "On Polynomial Factorization over Finite Fields," *Math Comp* **36**(153) pp 281-287

[Helfrich85] B Helfrich, "Algorithms to Construct Minkowski Reduced and Hermite

Reduced Lattice Bases," *Theor Comp Sci* **41** pp 125-139

[vdH&L85] M-P van der Hulst and A K Lenstra, "Factorization of Polynomials by Transcendental Evaluation," *Proc EUROCAL 85* (Linz) Springer LNCS 204 pp 138-145

[Kaltofen82] E Kaltofen, "Factorization of Polynomials," *Computing, Suppl* **4** pp 95-113

[Kaltofen83] E Kaltofen, "On the Complexity of Finding Short Vectors in Integer Lattices," *Proc EUROCAL 83* (London) Springer LNCS 162 pp 236-244

[Kaltofen84] E Kaltofen, "Effective Hilbert Irreducibility," *Proc EUROSAM 84* Springer LNCS 174 pp 277-284

[Kaltofen85a] E Kaltofen, "Sparse Hensel Lifting," *Proc EUROCAL 85* (Linz) Springer LNCS 204 pp 4-17

[Kaltofen85b] E Kaltofen, "Polynomial Time Reductions from Multivariate to Bi- and Univariate Integer Polynomial Factorization," *SIAM J Comp* **14** (May 1985)

[Kaltofen86] E kaltofen, "Polynomial Factorization 1982-1986," presented at *Computers and Mathematics* at Stanford University (Aug 1986)

[KMS83] E Kaltofen, D R Musser and B D Saunders, "A Generalized Class of Polynomials that are Hard to Factor," *SIAM J Comp* **12** (Aug 1983) pp 473-483; see also *Proc SYMSAC 81* (Snowbird) pp 188-194

[Knuth81] D E Knuth, "Seminumerical Algorithms," Addison-Wesley 1981 (2nd ed)

[K&T77] H T Kung and D M Tong, "Fast Algorithms for Partial Fraction Decomposition," *SIAM J Comp* **6** (Sep 1977) pp 582-593

[Landau85] S Landau, "Factoring Polynomials over Algebraic Number Fields," *SIAM J Comp* **14** (Feb 1985) pp 184-195

[L&McC87] L Langemyr and S McCallum, "The Computation of Polynomial Greatest Common Divisors over an Algebraic Number Field," preprint to appear in *Proc EUROCAL 87* (Leipzig)

[Lauer83] M Lauer, "Generalized *p*-adic Constructions," *SIAM J Comp* **12** (May 1983) pp 395-410

[Lazard82] D Lazard, "On Polynomial Factorization," *Proc EUROCAM 82* (Marseille) Springer LNCS 144 pp 126-134

[Lenstra82a] A K Lenstra, "Lattices and Factorization of Polynomials over Algebraic Number Fields," *Proc EUROCAM 82* (Marseille) Springer LNCS 144 pp 32-39; see also A K Lenstra, "Lattices and Factorization of Polynomials," *SIGSAM Bulletin* **15**(3) (Aug 1981) pp 15-16

[Lenstra82b] A K Lenstra, "Factorization of Polynomials," *Computational Methods in Number Theory* I (Mathematical Centre Tract 154) Mathematisch Centrum, Amsterdam 1982 (eds Lenstra & Tijdeman)

[Lenstra83a] A K Lenstra, "Factoring Multivariate Polynomials over Finite Fields," *Proc 15th Symp Th of Comp 1983* pp 189-192

[Lenstra83b] A K Lenstra, "Factoring Polynomials over Algebraic Number Fields," *Proc EUROCAL 83* (London) Springer LNCS 162 pp 245-254

[Lenstra84] A K Lenstra, "Polynomial Factorization by Root Approximation," *Proc EUROSAM 84* (Cambridge) Springer LNCS 174 pp 272-276

[Lenstra87] A K Lenstra, "Factoring Multivariate Polynomials over Algebraic Number Fields," *SIAM J. Comp* **16** pp 591-598

[LLL82] A K Lenstra, H W Lenstra and L Lovász, "Factoring Polynomials with Rational Coefficients," *Math Ann* **261** pp 515-534

[Lucks86] M Lucks, "A Fast Implementation of Multivariate Polynomial Factorization," *Proc SYMSAC 86* (Waterloo) pp 228-232

[Lugiez84] D Lugiez, "A New Lifting Process for the Multivariate Polynomial Factorization," *Proc EUROSAM 84* (Cambridge) Springer LNCS 174 pp 297-309

[Lugiez85] D Lugiez, "Fast Hensel Lifting Implementation," *Discr Math* **56** pp 214-225

[McEliece69] R J McEliece, "Factorization Of Polynomials over Finite Fields," *Math Comp* **23** pp 861-867

[Mignotte74] M Mignotte, "An Inequality about Factors of Polynomials," *Math Comp* **28**(128) pp 1153-1157

[Mignotte76] M Mignotte, "Some Problems about Polynomials," *Proc SYMSAC 76* pp 227-228

[Mignotte80] M Mignotte, "Factorization of Univariate Polynomials: a statistical study," *SIGSAM Bulletin* **14**(4) (Nov 1980) pp 41-44

[Mignotte81] M Mignotte, "Some Inequalities about Univariate Polynomials," *Proc SYMSAC 81* (Snowbird) pp 195-199

[Mignotte82] M Mignotte, "Some Useful Bounds," *Computing, Suppl* **4** pp 259-263 eds Buchberger, Collins, Loos (Springer-Verlag)

[M&Y74] A Miola & D Y Y Yun, "The Computational Aspects of Hensel-type Univariate Polynomial Greatest Common Divisor Algorithms," *Proc EUROSAM 74* pp 46-54

[Moenck77] R T Moenck, "On the Efficiency of Algorithms for Polynomial Factoring," *Math Comp* **31**(137) pp 235-250

[M&N81] P M A Moore and A C Norman, "Implementing a Polynomial Factorisation and GCD Package," *Proc SYMSAC 81* (Snowbird) pp 109-116

[Musser71] D R Musser, "Algorithms for Polynomial Factorization," PhD thesis (Tech Rep 134, Comp Sci Dept) Univ of Wisconsin, Sep 1971

[Musser75] D R Musser, "Multivariate Polynomial Factorization," *Journal ACM* **22** (Apr 1975) pp 291-308

[Musser78] D R Musser, "On the Efficiency of a Polynomial Irreducibility Test," *Journal ACM* **25** (Apr 1978) pp 271-282

[Rabin80] M O Rabin, "Probabilistic Algorithms in Finite Fields," *SIAM J Comp* **9** (May 1980) pp 273-280

[Rothstein77] M Rothstein, "A New Algorithm for the Integration of Exponential and Logarithmic Functions," *Proc 1977 MACSYMA Users' Conference* pp 263-274; and also see his PhD thesis "Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions," Univ of Wisconsin, 1976.

[Schnorr85] C P Schnorr, "A More Efficient Algorithm for Lattice Basis Reduction," *preprint for Proc ICALP 86*

[Schnorr86] C P Schnorr, "A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms," *preprint for Theory of Algebra, Coll Math Soc Janos Bolyai* **44** (publ North Holland)

[Schönhage84] A Schönhage, "Factorization of Univariate Integer Polynomials by Diophantine Approximation and an Improved Basis Reduction Algorithm," *preprint to appear in Proc ICALP 84* (Antwerpen)

[Trager76] B M Trager, "Algebraic Factoring and Rational Function Integration," *Proc SYMSAC 76* pp 219-226

[Vallée87] B Vallée, "An Affine Point of View on Minima Finding in Integer Lattices of Lower Dimensions," *to appear in Proc EUROCAL 87*

[Viry85] G Viry, "Polynomial Factorization over $\mathbb{Z}[x]$," *Proc AAECC-3* (Grenoble) Springer LNCS 229 pp 326-332

[Wang75] P S Wang, "Factoring Multivariate Polynomials over the Integers," *Math Comp* **29**(131) pp 935-950

[Wang76] P S Wang, "Factoring Multivariate Polynomials over Algebraic Number Fields," *Math Comp* **30**(134) pp 324-336

[Wang77] P S Wang, "Preserving Sparseness in Multivariate Polynomial Factorization," *Proc 1977 MACSYMA Users' Conf* pp 55-64; see also [Wang78], and P S Wang, "Factoring Larger Multivariate Polynomials," *SIGSAM Bulletin* **10**(4) (Nov 1976) p 42

[Wang78] P S Wang, "An Improved Multivariate Polynomial Factoring Algorithm," *Math Comp* **32**(144) pp 1215-1231; see also [Wang77], and P S Wang, "The EEZ-GCD Algorithm," *SIGSAM Bulletin* **14**(2) (May 1980) pp 50-60

[Wang79a] P S Wang, "Analysis of the *p*-adic Construction of Multivariate Correction

Coefficients in Polynomial Factorization: Iteration vs Recursion," *Proc EUROSAM 79* (Marseille) Springer LNCS 72 pp 291-300

[Wang79b] P S Wang, "Parallel *p*-adic Constructions in the Univariate Polynomial Factoring Algorithm," *Proc 1979 MACSYMA Users' Conf* pp 310-318

[Wang83] P S Wang, "Early Detection of True Factors in Univariate Polynomial Factorization," *Proc EUROCAL 83* (London) Springer LNCS 162 pp 225-235

[WGD82] P S Wang, M J T Guy and J H Davenport, "P-adic Reconstruction of Rational Numbers," *SIGSAM Bulletin,* (2) (May 1982).

[Wang&Rothschild75] P S Wang and L P Rothschild, "Factoring Multivariate Polynomials over the Integers," *Math Comp* **29**(131) pp 935-950

[Wang&Trager79] P S Wang and B M Trager, "New Algorithms for Polynomial Square-free Decomposition over the Integers," *SIAM J Comp* **8** pp 300-305

[Weinberger&Rothschild76] P J Weinberger and L P Rothschild, "Factoring Polynomials over Algebraic Number Fields," *ACM ToMS* **2** (Dec 1976) pp 335-350

[Wilkinson59] J H Wilkinson, "The Evaluation of the Zeros of Ill-conditioned Polynomials," *Num Math* **1** pp 150-180

[Zassenhaus69] H Zassenhaus, "On Hensel Factorization, I," *J N T* **1** pp 291-311

[Zassenhaus78] H Zassenhaus, "A Remark on the Hensel Factorization Method," *Math Comp* **32**(141) pp 287-292

[Zassenhaus81] H Zassenhaus, "Polynomial Time Factoring of Integral Polynomials," *SIGSAM Bulletin* **15**(2) (May 1981) pp 6-7

[Zippel79] R Zippel, "Probabilistic Algorithms for Sparse Polynomials," *Proc EUROSAM 79* (Marseille) Springer LNCS 72 pp 216-226; see also *Proc 1979 MACSYMA Users' Conf* pp 308-309

[Zippel81] R Zippel, "Newton's Iteration and the Sparse Hensel Algorithm," *Proc SYMSAC 81* (Snowbird) pp 68-72

Appendix A. Notation

Symbol	Meaning
$\mathbb{Z}$	the ring of integers
$\mathbb{Q}$	the field of rationals
$\mathbb{R}$	the field of real numbers
$\mathbb{C}$	the field of complex numbers
$\mathbb{A}$	the field of algebraic numbers
$\mathbb{F}_q$	the finite field of size q
$\mathbb{Q}(\alpha)$	the extension of $\mathbb{Q}$ generated by α
O_K	the ring of integers in the algebraic number field K
$\mathbb{Z}^n$	the vector space over $\mathbb{Z}$ of dimension n
$\underline{e}_j$	the unit vector in the j^{th} direction (chapter 5 page 2)
∂f	the degree of f
$\partial_x f$	the degree of f in x
$\partial\alpha$	the degree of the algebraic number α over the obvious field
$\partial_K\alpha$	the degree of α over the field K
$gcd(f, g)$	the greatest common divisor of f and g
$det(M)$	the determinant of M
$\lfloor x \rfloor$	the largest integer not exceeding x
$\lceil x \rceil$	the smallest integer not less than x
$\{x\}$	the integer closest to x, rounded down in ambiguous cases
$\\| f \\|_\infty$	height of $f := \sum a_i x^i$, i.e. $\max\{\|a_0\|, \ldots, \|a_n\|\}$
$\\| f \\|_m$	the m-norm of f, i.e. $(\sum_{i=0}^{n} \|a_i\|^m)^{1/m}$
$\\| \alpha \\|$	the maximum of the absolute values of the *field* conjugates of α
(a, b), $(a, b]$, $[a, b)$, $[a, b]$	subintervals of the real line
(b_j^*, b_k)	inner product of vectors b_j^* and b_k (only in chapter 5)
$\|b_j\|$	Euclidean length of the vector b_j (chapter 5)
$R<b_1, \ldots, b_n>$	the R-module generated by $b_1, \ldots, b_n$
$O(f(n))$	functions bounded above by a fixed multiple of $f(n)$ for large n
$\Omega(f(n))$	functions bounded below by a fixed multiple of $f(n)$ for large n
$(x_2-a_2, \ldots, x_n-a_n)$	ideal generated by the polynomials x_2-a_2 through x_n-a_n
$(x_2-a_2, \ldots, x_n-a_n)^k$	k^{th} power of the ideal
$GL_d(\mathbb{Z})$	the set of $d \times d$ invertible matrices with integer entries

Appendix B. Glossary

This is a list of brief definitions of terms as used in this thesis. Fuller definitions are in chapter 2, "Fundamentals and Definitions". The definitions are alphabetically ordered (except for the first two).

Phrase	**Definition**
$\mathbb{Q}$-basis	A is a $\mathbb{Q}$-basis for B if every element of B is a sum of rational multiples of elements of A
$\mathbb{Z}$-basis	A is a $\mathbb{Z}$-basis for B if every element of B is a sum of integer multiples of elements of A
algebraic closure	of a field F is a field $\overline{F}$ containing all roots of all polynomials over F
algebraic function	a root of a multivariate polynomial (rational numbers excluded)
algebraic integer	a root of a monic polynomial with non-fractional coefficients
algebraic kernel	a symbol in the computer for an algebraic integer
algebraic number	a root of a univariate polynomial (rational numbers excluded)
BANP	the Bath Algebraic Number Package (also handles algebraic functions)
conjugate	*see field conjugate*
defect	the biggest denominator in the representation of any algebraic integer

degree	of an algebraic symbol is the degree of its minimal polynomial
degree	of an extension field is the product of the degrees of the generators
discriminant	the square of the determinant of the basis elements and all their field conjugates
discriminant	of a polynomial f is *resultant*(f, f')
embedding	a map allowing one field to be regarded as a subset of another
extension	$F(\alpha)$ where F is a field and $\exists f(x) \in F[x] : f(\alpha)=0$
field conjugate	one of the images under an embedding into an algebraic closure
fundamental region	for a basis $\underline{b}_1, \ldots, \underline{b}_n$ is the set of points $\{\beta_1\underline{b}_1 + \cdots + \beta_n\underline{b}_n : -\frac{1}{2} < \beta_i \le \frac{1}{2}\}$
Hensel lifting	a way of obtaining solutions mod p^n from one mod p
integral basis	a $\mathbb{Z}$-basis for the ring of algebraic integers
Kronecker's trick	substituting high powers of one variable for all the others
lifting	see Hensel lifting
minimal polynomial	of α is the monic polynomial of least degree having α as a root
monic	having leading coefficient 1
multivariate	having several variables
norm	product of all the field conjugates
obvious basis	for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ is $\{\alpha_1^{e_1} \cdots \alpha_n^{e_n} : 0 \le e_i < \partial\alpha_i\}$
orthogonality defect	ratio of the product of the lengths of the basis vectors to their determinant
primitive element	an element of a field which will generate the field on its own

simple extension	an extension by a single algebraic element
Swinnerton-Dyer	polynomials factorise into linears and quadratics modulo all primes
tower	a sequence of simple extensions
univariate	having only one variable

Appendix C. Program to Test a Conjecture

Below is a listing of the FORTRAN 77 program used to test the conjecture in chapter 4 (on bounds). We ran the program with input values for *size* being successively 1, 2, 4, 8 16, 32, 64, and 128 for each value of *degree* going from 3 to 9 inclusive. The complete absence of output lends considerable support to the conjecture.

The Program

```
C       This is a FORTRAN program to test a conjecture in my thesis.
        complex a(10), b(10), shift, coeff, eye
        integer i, j, k, degree, choose
        real size, rb

C       Set the variable 'eye' to be the complex number i.
        eye = (0.0, 1.0)
C       Ask for the degree of the polynomials to be tested, and also
C       for a bound on their height.
        print*, "Enter degree and height bound"
        read*, degree, size

C       Now do 1000 random tests of polynomials of degree 'degree' and
```

```
C       height at most 'size'
        do 999 k = 1, 1000
        do 10 i = 1, degree
10              a(i) = ((2.0*rand(0)-1.0)*size)*eye + ((2.0*rand(0)-1.0)*size)
        a(degree+1) = 1
C       The complex array 'a' now holds a monic polynomial
C       sum from i=1 to degree+1 (a(i)*x**(i-1)).

        shift = -a(degree)/float(degree)
        do 20 i = 1, degree+1
                coeff = a(i)
                do 50 j = i+1, degree+1
50                      coeff=coeff + a(j)*choose(j-1,i-1)*shift**(j-i)
20              b(i) = coeff
C       The complex array 'b' holds the monic polynomial derived from 'a'
C       by making the linear substitution x -> x-a(degree)/degree
C       which causes the term in x**(degree-1) to vanish.

C       The conjecture is that rb(a) >= rb(b)+absolute_value(shift) always,
C       where rb(a) and rb(b) represent the values calculated in the next
C       two lines
        rba = rb(a, degree)
        rbb = rb(b, degree)
C       "fudge factor" in the line below as rb may be too large by a
C       factor of 1.0001 -- see the code for rb(). Print only if the conjecture fails.
        if (rbb + cabs(shift) .ge. 1.0001*rba) print*, (rbb + cabs(shift))/rba, a
999     continue
        end
```

```
C     This function takes a polynomial of degree 'degree' held in 'poly'
C     as sum from i=1 to degree+1 (poly(i)*x**(i-1)).
C     The result is a real number close to the largest (and sole positive)
C     real root of the derived polynomial:
C     x**degree - sum from i=1 to degree (cabs(poly(i))*x**(i-1))
C     where cabs() is the complex absolute value function

      real function rb(poly, degree)
      complex poly(10)
      integer degree
      real low, high, mid, val

C     We assume the positive root lies between 0 and 9999, and search for
C     it using a simple binary chop method.
      low = 0.0
      high = 9999.0
40    if (high - low .lt. 0.0001*high) go to 987
      mid = (low+high)/2
      val = mid**degree
      do 30 i = 1, degree
30          val = val - cabs(poly(i))*mid**(i-1)
      if (val .gt. 0.0) then
            high = mid
      else
            low = mid
      endif
      go to 40
987   rb = high
```

```
      end

C     Below is the standard combinatorial 'choose' function
      integer function choose(n, r)
      integer answer, i, r, n

      answer = 1
      if (n-r .lt. r) r = n-r
      do 123 i = 1, r
123       answer = (answer * (n-i+1))/i
      choose = answer
      end
```

Appendix D. Lenstra's Examples

At many points during this thesis the five examples given in [Lenstra82] are used as test cases. The examples he gave are listed below in the order they appeared in the original paper. In each case f is the polynomial to be factorized and m is the minimal polynomial of the extension generator.

Example 1

$$f(x) = \frac{1}{47}(47x^6+21x^5+598x^4+1561x^3+1198x^2+261x+47)$$

$$m(\alpha) = \alpha^2-\alpha+3$$

Example 2

$$f = \frac{1}{16}(16x^6-1)$$

$$m(\alpha) = \alpha^3+2$$

Example 3

$$f(x) = x^8-x^7-x^6+x^4-x^2+x+1$$

$$m(\alpha) = \alpha^4-\alpha+1$$

Example 4

$f(x) = x^3-3$

$m(\alpha) = \alpha^6+3\alpha^5+6\alpha^4+\alpha^3-3\alpha^2+12\alpha+16$

Example 5

$f(x) = x^9+9x^8+36x^7+69x^6+36x^5-99x^4-303x^3-450x^2-342x-226$

$m(\alpha) = \alpha^9-15\alpha^6-87\alpha^3-125$

Appendix F. Trager vs Lenstra

The table below compares our implementations of the factorization algorithms in [Trager76] and [Lenstra82] on the examples given in [Lenstra82]. The entries in the table are times in seconds for the complete factorization under REDUCE 3.3.

Trager versus Lenstra		
Example	Trager	Lenstra
1	155	10.4
2	14.0	8.4
3	2720	39
4	299	92
5	>3000	2430

Notice that there is a very great variation in the ratios of the times. In our experience, the "simpler" factorizations take about the same amount of time whereas more difficult problems increasingly favour Lenstra's algorithm — the figures in the table bear this out after allowing for fluctuations in the running times.

Index.