


POLYNOMIAL FACTORIZATION*

Vilmar Trevisan 

Abstract

It is perhaps surprising that an algorithm for polynomial factorization is so sophisticated and complex, representing one of the major successes of the Symbolic and Algebraic Computation. In this report we study some details of the algorithms to perform this task, which has applications in many areas of mathematics.

1 Introduction

Polynomials play a very important role in many areas such as Numerical, Real and Complex Analysis, Number Theory, Approximation Theory and Control Theory, just to mention a few areas in mathematics. In Symbolic Computation, polynomials are important not only because algorithms based on polynomials form a solid foundation of any symbolic computation system, but also because they pose new challenging mathematical problems.

Consider, for example, the integral

$$\int \frac{dx}{x^2 + x - 12}.$$

The best known integration algorithms would first *factor* $x^2 + x - 12$ as $(x - 3)(x + 4)$. Based on this factorization, the integrand is decomposed into simpler expressions and the integral computed. Specifically,

$$\begin{aligned} \int \frac{dx}{x^2 + x - 12} &= \int \left(\frac{1/7}{x - 3} - \frac{1/7}{x + 4} \right) dx \\ &= \frac{\log(x - 3)}{7} - \frac{\log(x + 4)}{7} + C. \end{aligned}$$

*This is the report of a talk given at the XII Brazilian School of Algebra, which took place in Diamantina, in August, 1992

The generalization of this calculus-homework technique results in an extremely complex mathematical problem, whose study led to one of the major successes of computer algebra, namely, the Risch integration algorithm.

Nontrivial integration problems would involve factorization of *multivariate* polynomials over *algebraic extensions* of the rational numbers. However, even in these more general settings, the factorization is either reduced to or is heavily dependent on factoring univariate polynomials. Other applications of polynomial factorization can be found in the areas of coding theory, theory of control, and Galois theory, just to cite a few.

1.1 Previous Work

The problem of factoring polynomials has a prominent history that D. Knuth [9] and E. Kaltofen [6] trace back to Newton's *Arithmetica Universalis* (1707), where an algorithm to find linear and quadratic factors of a polynomial with integer coefficients is presented. This method was extended by the astronomer F. Schubert in 1793 who devised an algorithm to find all the factors of an integral polynomial of degree n . L. Kronecker rediscovered the method in 1882 and also gave algorithms for factoring polynomials with two or more variables, whose ideas are still used for factorization over some domains.

The basic idea of the so called Kronecker algorithm is the *evaluation-interpolation* technique. Recall that a polynomial of degree n is completely determined by its values at $n + 1$ different points. A factor $h(x)$ of degree d is determined by its values at $d + 1$ points, say $h(x_1), \dots, h(x_{d+1})$. Also the values of each $h(x_j)$ must be integer divisors of $f(x_j)$. So the integers $f(x_j)$ are factorized and we try to interpolate from all possible combinations of the divisors of $f(x_1), \dots, f(x_{d+1})$. If we find a factor we divide it out and change the $f(x_i)$ appropriately. The irreducibility of the factors is ensured by first locating the degree 1 factors, then degree 2 and so on.

This algorithm turned out to be very inefficient. It solves the factorization of polynomials by factoring integers, a problem that can be much harder. Major advances in computer algebra during the last two decades allow the factoriza-

$$\begin{array}{ccc}
 f(x) \in \mathbb{Z}[x] & & f(x) = f_1(x) \cdots f_r(x) \\
 \downarrow \text{mod } p & & \uparrow \\
 f_p(x) \text{ mod } p & \longrightarrow f_p(x) = f_p^1(x) \cdots f_p^k(x) \text{ mod } p & \longrightarrow f_q(x) = f_q^1(x) \cdots f_q^k(x) \text{ mod } q
 \end{array}$$

Figure 1: Factorization over the integers

tion of polynomials encountered in practice to be performed very quickly. For example a degree 40 polynomial shown in [16] takes only seconds to factor.

A landmark in this direction is a very efficient algorithm for factoring polynomials over \mathbb{Z}_p , the field of integers modulo a prime p , introduced in 1967 by E. Berlekamp [1]. The efficiency of the algorithm suggested factoring an integral polynomial by first factoring modulo small primes and then reconstructing the integral factorization by some means such as the Chinese Remainder Theorem.

This technique, known as the *modular method* has been used to solve a large class of problems in computer algebra. Examples include the computation of polynomial greatest common divisors, big number arithmetic, partial fraction decomposition and a whole class of matrix calculations, such as determinants, inverses, and solution of system of linear equations. The Chinese Remainder Theorem, however, does not apply to the polynomial factorization problem.

Another milestone in the factoring process was the 1969 paper by H. Zassenhaus [18]. Based on a result of 1918 called "Hensel's Lemma", Zassenhaus shows how to "lift" a factorization from modulo p^k to modulo p^{2k} . The integer-modular-integer relationship was then complete. After the integral polynomial f is factored over \mathbb{Z}_p , the factorization modulo p is lifted to a factorization modulo q , q exceeding an estimate on the size of the largest coefficient of any factor of f . From this factorization modulo q , the true factorization over the integers can be recovered via combinations of one or more mod q factors.

The diagram of figure 1 illustrates pictorially the structure of modern algorithms for factoring polynomials over the integers.

D. Musser and P. Wang & L. Rothschild extended the integer-modular relationship to factorization of multivariate polynomials over the integers. The

extension to factorization over algebraic number fields was made by P. Wang in 1976. In 1982, A. Lenstra, H. Lenstra & Lovász [10] introduced a polynomial time complexity algorithm, named LLL, for recovering the true factorization over the integers given the lifted univariate factorization. This shows that univariate polynomial factorization can be done in polynomial time. Finally, a result by E. Kaltofen [7], allows the reduction of multivariate to univariate integral polynomial factorization to be done in polynomial time. For a complete survey on polynomial factorization, the papers [6, 8] are recommended.

We present here some aspects of univariate factorization over finite fields and refer to [16] for the factorization over the integers.

2 Factoring over Finite Fields

We consider here the factorization of a polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \quad a_n \neq 0$$

with coefficient $a_i \in GF(q)$, the Galois field of $q = p^k$ elements, for p a prime number and k a positive integer, into its irreducible factors in $GF(q)$. We assume the polynomial f is *monic* and *squarefree*, that is its leading coefficient is 1 and that it does not have repeated factors.

Finally, we present the well known Berlekamp algorithm, which is the prototype algorithm for factorization over small finite fields. In section 4, we introduce a result that enables us to reduce the complexity of factorizations over extensions of finite fields. We then present the Cantor-Zassenhaus algorithm that is more efficient than Berlekamp's for larger values of p .

In section 6, we discuss an efficient implementation of the Distinct Degree Factorization, a partial factorization that, in principle, speeds up all algorithms.

3 Overview of Berlekamp's Algorithm

Berlekamp's algorithm [1] is a major milestone in the study of the polynomial factoring problem, and a brief review is pertinent here. Our presentation is for

$GF(p)$, even though the procedure is also valid for $GF(p^k)$.

Assuming that $f(x)$ is monic and squarefree in $GF(p)[x]$, where p is a prime, Berlekamp's algorithm was devised based on the following observations:

1. The r irreducible factors $f_1(x), f_2(x), \dots, f_r(x)$ of $f(x)$ are relatively prime in $GF(p)[x]$. Given a set of residues s_i in $GF(p)$ and using the Chinese Remainder Theorem, there is a unique polynomial $v(x) \in GF(p)[x]$, such that:

$$\begin{aligned} \text{(a)} \quad & \deg(v) < \deg(\prod_{i=1}^r f_i) = \deg(f) \\ \text{(b)} \quad & v(x) = s_i \pmod{f_i(x)}, \quad 1 \leq i \leq r. \end{aligned} \tag{1}$$

This implies that given $v(x)$ and the residues s_i , we can compute the factors $f_i(x)$ by taking gcds

$$f_i(x) = \gcd(v(x) - s_i, f(x)).$$

2. Applying the result given by

Lemma 3.1 *If $g(x)$ is a polynomial in $GF(q)[x]$, then $(g(x))^q = g(x^q)$.*

to equation (1) we have

$$v(x)^p = s_i^p = s_i = v(x) \pmod{f_i(x)}.$$

Because of the above Lemma and because $f_i(x) | f(x)$ it is sufficient to find a polynomial $v(x)$ such that

$$v(x^p) - v(x) = 0 \pmod{f(x)}.$$

3. In order to find the polynomials $v(x)$ as above, we can use matrix operations. Consider the matrix Q (see section 6). Then the solutions $v(x)$ of the equation above can be viewed as solutions of

$$\bar{V}[Q - I] = \bar{0} \tag{2}$$

where \bar{V} is the vector of coefficients of $v(x)$. In other words, the problem reduces to finding the null space of the system of equations (2).

4. The number r of solutions $v(x)$ of (2) is the number of factors of $f(x)$.

Based on these observations, we can factor a monic squarefree polynomial $f(x)$ according to

Algorithm 3.1 BER

Step 1. Fill the matrix Q

Step 2. Find the null space of $\bar{V}[Q - I] = \bar{0}$

Step 3. Compute $w(x) = \gcd(v(x) - s, f(x))$ for $0 \leq s < p$
and for some $v(x)$ a solution of $\bar{V}[Q - I] = \bar{0}$.

Step 4. Step 3 produces at least one non-trivial factor of $f(x)$. If the number of factors is $< r$, then repeat Step 3 with a different $v(x)$, otherwise terminate.

This algorithm is very efficient for small values of p . Its worst case complexity is

$$O(n^3 + rpn^2) \tag{3}$$

operations in $GF(p)$ where r is the number of irreducible factors and n is the degree of $f(x)$ [5, 9]. The expression above is an upper bound for the running time of Berlekamp's algorithm and it shows that for larger values of p this is not a good approach to factor $f(x)$ over $GF(p)$.

4 Factorization over Large Finite Fields

Algorithms like Berlekamp's where we try all possible values of $GF(q)$, are called *deterministic*. Notice that, necessarily, the running time of such a procedure is proportional to q , which is not desirable for large q .

For factoring polynomials over large finite fields, Berlekamp [2] devised a different algorithm. It aides the factorization problem by finding roots of a polynomial with coefficients in the prime field. The best running time algorithm [6] is due to R. Moenk and it is in $O(n^3 + n^2 \log p + n \log p)$ if $p - 1$ is highly composite. The number of polynomial gcd's required in Berlekamp's algorithm is proportional to q . When q is large it becomes advantageous to find the s 's

such that $\gcd(f(x), v_i(x) - s) \neq 1$. This can be done by computing the residues modulo $f(x)$ of $1, v_i(x), v_i^2(x), \dots$ until a power of $v_i(x)$ is a linear combination of the previous powers. It can be shown that $1, v_i(x), v_i^2(x), \dots, v_i^r(x)$ are always linearly dependent modulo $f(x)$. This linear combination can be written as a polynomial $G_i(v_i(x)) = \sum_{j=0}^r g_j v_i^j(x)$. It can be shown [18] that

$$G_i(v_i(x)) = \prod (v_i(x) - s), \quad (4)$$

with $\gcd(f(x), v_i - s)$ nontrivial.

Hence $G_i(v_i(x))$ splits and its roots are the desired s values. Thus the problem of finding factors of f is reduced to that of finding the roots of $G_i(v_i(x))$ in $GF(q)$, with $q = p^k$. The roots of a polynomial $G(x) \in GF(q)[x]$ can be computed using an algorithm of Berlekamp [2]. Three different approaches apply depending on the type of the field: for $q = p$ large, q large but p small, or otherwise.

Algorithms of this nature are said to be *probabilistic*, since the elements are chosen at random and, after a suitable transformation to increasing probability, are tested for roots.

The Cantor-Zassenhaus algorithm, discussed in detail below, can be applied to the factorization problem in any finite field. The time complexity is polynomial in n , the degree of the polynomial, and $\log q$, q being the cardinality of the field, which makes the procedure more efficient than Berlekamp's. The drawback, however, is that these algorithms have to perform arithmetic intensively in the field, which is quite costly if it is an extension of Z_p .

The most common factoring problem is when a polynomial $f(x)$ with coefficients in Z_p is to be factored over an extension $GF(p^k)$. For this case we can speed up the Cantor-Zassenhaus algorithm by first factoring f over Z_p . There still remains the task of factoring over $GF(p^k)$ the factors of f that are irreducible in $Z_p[x]$. The following result gives, without any cost, the number of irreducible factors and its respective degrees in $GF(p^k)$.

Theorem 4.1 *Let f be an irreducible polynomial of degree n in a finite field*

$GF(q)$. If k is natural number and $d = \gcd(n, k)$ then f has d irreducible factors over $GF(q^k)$ all of degree n/d .

Proof. The order of the irreducible polynomial f is the order of any root of f in the multiplicative group $GF^*(q^n)$. Recall that $GF(q^n)$ is the splitting field of f hence it contains all roots of f . Let g be an irreducible factor of f in $GF(q^k)$. If e is the order of f , then the order of g is also e . The multiplicative order of q considered as an element of the ring R , the integers modulo e is n and the degree of g is equal to the multiplicative order of q^k in R (see [11]). The powers $q^j, j = 0, 1, \dots$, considered modulo e , form a cyclic group of order n . Thus the order of q^k is n/d and so is the degree of g .

Observe that when n and k are relatively prime, the factor is already irreducible over the extension and there is no need to further break it. If the factor is reducible in $GF(p^k)$ we proceed with the Cantor-Zassenhaus algorithm. The arithmetic will be done now in the extension field. This theorem is bound to speed up the factorization for two reasons. Firstly it avoids as much as possible the arithmetic in the field $GF(q^k)$. Secondly, it deals with smaller problems.

5 Overview of Cantor-Zassenhaus Algorithm

In 1981, Cantor and Zassenhaus [4] introduced a new probabilistic factorization algorithm whose expected running time is polynomial in n , the degree of f , and $\log q$, q being the cardinality of the field $GF(q)$, making it suitable for large values of q . We give here a brief outline.

Let $q = p^k$ be the size of the finite field $GF(q)$, with p a prime and k a positive integer. The polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0,$$

with the coefficients $a_i \in GF(q)$, is to be factored into irreducible polynomials over $GF(q)$. We can use the Cantor-Zassenhaus [4] algorithm to carry out the factorization.

Algorithm 5.1 : CZ

CZ-1. Make sure that f is squarefree and is monic.

CZ-2. Factor f into a product

$$f(x) = \prod_{i=1}^n h_i(x), \quad (5)$$

where each $h_i(x)$ contains only irreducible factors of degree i

CZ-3. Factor each $h_i(x)$ into irreducible factors.

Step CZ-1 is the same as in the Berlekamp's algorithm. The partial factorization of Step CZ-2 is called the *distinct degree factorization*, and will be discussed in detail presently. Many ways to factor $h_i(x)$ are discussed in [15]. Below we present the Cantor-Zassenhaus original technique. We call Step CZ-3 uniform degree factorization .

6 Distinct Degree Factorization

A basic algorithm to perform CZ-2 is based on the following

Lemma 6.1 *The polynomial $x^{q^m} - x$ is the product of all distinct monic irreducible polynomials in $GF(q)[x]$ with a degree dividing m .*

and seems to be known to S. Schwarz as early as 1956.

Factorization (5) will be achieved using the Distinct Degree Factorization (DDF) algorithm below. Most h_i will be 1, but the factorization will be non-trivial unless all factors of f have the same degree. In any case, DDF reveals the number and the degrees of the irreducible factors of f .

Algorithm 6.1 : DDF

DDF-1. Set $i \leftarrow 1$, $r_0(x) \leftarrow x$

DDF-2. Set $r_i(x) \leftarrow (r_{i-1}(x))^q \bmod f(x)$

DDF-3. Set $h_i(x) \leftarrow \gcd(f(x), r_i(x) - x)$

If $h_i \neq 1$, set $f(x) \leftarrow f(x)/h_i(x)$.

If $f(x) = 1$ terminate

DDF-4. Set $i \leftarrow i + 1$.

If $2i \leq \deg(f(x))$ go to **DDF-2**,

else set $h_{\deg(f)} \leftarrow f(x)$ and terminate.

The costliest step is, clearly, DDF-2. The usual way to raise $r_i(x)$ to a large power is to use the *binary method* [9]. The number of multiplications modulo $f(x)$ is $O(\log q)^1$. Based on a suggestion by Berlekamp [2] we recommend the following different approach.

Suppose $r_{i-1}(x) = b_{n-1}x^{n-1} + \dots + b_1x + b_0$ is a polynomial of $GF(q)[x]$, then, from Lemma 3.1, $r_i(x) = (r_{i-1}(x))^q = b_{n-1}x^{q(n-1)} + \dots + b_1x^q + b_0$. Therefore, if we pre-compute the values

$$Q_i(x) = x^{iq} \bmod f(x), \text{ for } i = 0, \dots, n-1. \quad (6)$$

and store Q_i as the i th row of the $n \times n$ matrix Q , we have

$$r_i(x) = r_{i-1}(x) \cdot Q. \quad (7)$$

Once we have formed the matrix Q , equation (7) provides a fast way to compute r_i from r_{i-1} . If we have to compute various $r_i(x) \bmod f(x)$, where $f(x)$ does not change (when f is irreducible for example), the total savings more than justify these pre-computation steps. In [15] can be found details on how to fill the matrix Q .

An example. Consider the polynomial

$$\begin{aligned} F_3(x) = & 904050 x^7 + 1479450 x^6 - 2336817 x^5 - 3403088 x^4 \\ & + 2021847 x^3 + 1477766 x^2 - 1006566 x + 170694, \end{aligned}$$

to be factored modulo $p = q = 11$.

¹Here and throughout this report $\log(x)$ represents $\log_2(x)$

Notice that F_3 is squarefree over Z_{11} . In this case, $n(\log q + 2n - 5) + 2 = 96$, whereas $nq = 77$. The shift-add method [16] is advantageous to fill the matrix Q . The resulting Q matrix is

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -5 & -3 & -5 & 2 & 3 & 3 & -1 \\ -1 & -3 & -2 & 5 & 2 & -1 & -1 \\ 1 & -4 & 0 & 3 & 2 & -4 & -5 \\ -1 & -5 & 0 & -4 & 5 & -4 & 2 \\ -5 & 2 & 4 & -2 & -5 & -3 & 4 \\ 3 & 3 & 0 & 4 & 4 & 3 & 1 \end{bmatrix}$$

Notice that

$$F_3(x) \equiv f(x) = 4x^7 + 5x^6 + x^5 + 4x^4 + 3x^3 + 4x^2 - 4 \pmod{11}.$$

We now go through algorithm NDDF. Step NNDF-1 initializes i to 1 and $r(x)$ to the second row of Q , which means $r(x) = -5x^6 - 3x^5 - 5x^4 + 2x^3 + 3x^2 + 3x - 1$. Now NDDF-2 returns $h_1(x) = \gcd(r(x) - x, F_3(x)) = x^3 + x^2 + 3x + 5$, while NDDF-3 sets $f = 4x^4 + x^3 - x^2 + 4x - 3$. NDDF-4 sets $i = 2$ and $r(x) = x$. Then, step NDDF-2 sets $h_2 = 4x^4 + x^3 - x^2 + 4x - 3$, and the algorithm terminates. There are 3 irreducible factors of degree 1, whose product is

$$h_1 = x^3 + x^2 + 3x + 5,$$

and 2 irreducible factors of degree 2, whose product is

$$h_2 = 4x^4 + x^3 - x^2 + 4x - 3$$

7 Uniform Degree Factorization

After distinct degree factorization, we have reduced the problem of factoring the general polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

over $GF(q)$ to the problem of factoring a monic polynomial

$$h(x) = h_i(x) = x^l + d_{l-1} x^{l-1} + \cdots + d_1 x + d_0, \quad (8)$$

which is a product of irreducible factors of the same degree i . Factoring this polynomial is trivial if $l = i$.

A polynomial $t(x) \in GF(q)[x]$ is a *separating polynomial* for $h(x)$ if $0 < \deg(\gcd(t, h)) < l$. The uniform degree factorization problem, therefore, has been reduced to finding separating polynomials for $h(x)$. Next we will describe the Cantor-Zassenhaus method to solve this problem. Additional methods for the uniform degree factorization can be found in [15, 16].

Algorithm 7.1 : CZSEP

CZSEP-1. Choose a random polynomial $t(x)$ in $GF(q)[x]$ of degree $\leq 2i - 1$

CZSEP-2. Set $t(x) \leftarrow (t(x))^{\frac{q-1}{2}} \bmod h(x)$

CZSEP-3. Compute $g(x) = \gcd(h(x), t(x) - 1)$

As an example, take $h_2 = 4x^4 + x^3 - x^2 + 4x - 3$ from the example of section 6. We know it is a product of two irreducible factors of degree 2 over Z_{11} . Taking $t(x) = x + 1$, at random. In CZSEP-2 $t(x)$ is set to $t(x)^{60} \bmod h_2(x) = 3x^3 - 3x^2 - x + 1$. Now $\gcd(h_2(x), t(x) - 1) = x^2 - x - 4$. We conclude that, modulo 11,

$$4x^4 + x^3 - x^2 + 4x - 3 = 4(x^2 - x - 4)(x^2 + 4x + 5).$$

An example. Consider the polynomial

$$f(x) = 9x^9 + 3x^8 + 14x^7 - 16x^6 + 5x^5 - 11x^4 - 3x^3 - 2x^2 - 13x - 13,$$

with coefficients in Z_{37} , to be factored over the field $K = GF(37^3)$. Since $\gcd(f, f') = 1$, we conclude that f is squarefree in K . Firstly, our algorithm factors f over Z_{37} . For this, after a monic transformation, the DDF returns the partial factorization

$$\begin{aligned} f(x) &= h_1(x) && h_2(x) && h_4(x) \\ &= (x + 10) && (x^4 + 15x^3 + 16x^2 - 16x + 7) && (x^4 - 6x - 13), \end{aligned}$$

where h_1 and h_4 are irreducible and h_2 is product of 2 irreducible factors of degree 2. To further break h_2 , we use the Cantor-Zassenhaus separator and obtain

$$h_2(x) = (x^2 - 12x - 3)(x^2 - 10x + 10).$$

Using Theorem 4.1, we see that this is the factorization into irreducible factors over K .

For the factorization of integral polynomials, that is, the lifting of modular factors and the recovery of integral factors, we refer to [16].

References

- [1] E. R. Berlekamp, *Factoring Polynomials over Finite Fields*, Bell System Tech. J., V. 46, 1967, pp. 1853-1859.
- [2] E. R. Berlekamp, *Factoring polynomials over large finite fields*, Math. Comp., v. 24, 1970, pp. 713-735.
- [3] B. Buchberger, G. E. Collis, and R. Loos (editors) *Computer Algebra, Symbolic and Algebraic Computation*. Springer-Verlag, 1983, New York.
- [4] D. Cantor and H. Zassenhaus, *A New Algorithm for factoring polynomials over finite fields*, Math. Comp., 36 (1981), pp. 587-592.
- [5] J. Davenport, Y. Siret, E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press. 1988.
- [6] E. Kaltofen, *Factorization of Polynomials*, in *Computer Algebra-Symbolic and Algebraic Computation*, Computing, Suppl. 4, Springer-Verlag, 1982, pp. 95-113.
- [7] E. Kaltofen, *Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization*, SIAM J. Comp., Vol. 14, 1985, pp. 469-489.