Fast Polynomial Factorization Over High Algebraic Extensions of Finite Fields*

Erich Kaltofen

Department of Mathematics
North Carolina State University
Raleigh, North Carolina 27695-8205, USA
kaltofen@eos.ncsu.edu
http://www4.ncsu.edu/~kaltofen

Victor Shoup

IBM Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon, Switzerland
sho@zurich.ibm.com

Abstract

New algorithms are presented for factoring polynomials of degree n over the finite field of q elements, where q is a power of a fixed prime number. When $\log q = n^{1+a}$, where $a > 0$ is constant, these algorithms are asymptotically faster than previous known algorithms, the fastest of which required time $\Omega(n(\log q)^2)$,† or $\Omega(n^{3+2a})$ in this case, which corresponds to the cost of computing x^q modulo an n degree polynomial. The new algorithms factor an arbitrary polynomial in time $O(n^{3+a+o(1)} + n^{2.69+1.69a})$. All measures are in fixed precision operations, that is in bit complexity. Moreover, in the special case where all the irreducible factors have the same degree, the new algorithms run in time $O(n^{2.69+1.69a})$. In particular, one may test a polynomial for irreducibility in $O(n^{2.69+1.69a})$ bit operations. These results generalize to the case where $q = p^k$, where p is a small prime number relative to q.

1 Introduction

The expected running time of randomized algorithms for factoring a polynomial over a finite field is measured in both the degree of the polynomial, n, and the cardinality of the field, q. The cost of the field arithmetic itself depends on the data structure representing field elements. Usually, each field operation costs $O((\log q)^{1+o(1)})$ fixed precision integer operations. Which of the known algorithms performs best in the sense of asymptotic running time depends on the size q relative to the degree n. If $\log q = O(n^b)$ with $b < 0.454$ the algorithms by Kaltofen and Shoup [15] require $O(n^{1.815+0.408b})$ expected arithmetic operations in $\mathbb{F}_q$, the field with q elements. If $\log q = \Theta(n^c)$ with $0.454 \le c \le 1.375$ the algorithm by von zur Gathen and Shoup [10] uses $O((n^2 + n\log q)(\log n)^2 \operatorname{loglog} n)$ field operations. For any larger q the dominant complexity of any algorithm is $O(n^{1+o(1)}\log q)$ field operations, which arises from computation of x^q modulo the polynomial to be factored and which already the Berlekamp [3] algorithm achieves (see also [21, 8], and [6]).

Here we focus on the latter case when additionally $q = p^k$ with p prime and where $\mathbb{F}_q$ is represented in the Kronecker style as $\mathbb{F}_p[z]/(\varphi(z))$, where φ is an irreducible polynomial over $\mathbb{F}_p$ of degree k. For the sake of introduction, we shall set $k = O(n^{1+a})$, where $a \ge 0$ is constant, and $p = O(1)$. Arithmetic operations in $\mathbb{F}_q$ become polynomial operations of computational complexity $O(n^{1+a+o(1)})$. The $n^{o(1)}$ factor represents logarithmic factors arising in the FFT based polynomial multiplication algorithms [22, 7]. Using any of the algorithms stated above, the number of fixed precision integer operations for factoring a polynomial of degree n over $\mathbb{F}_q$ is then $O(n(\log q)^{2+o(1)})$, or, in terms of a, $O(n^{3+2a+o(1)})$.

In this paper we show that under the described circumstances the fixed precision complexity of computing all irreducible factors of a polynomial of degree n over $\mathbb{F}_q$ is $O(n^{3+a+o(1)} + n^{2.69+1.69a})$. Our new algorithm uses randomization and the measure is the expected number of fixed precision operations. Furthermore, we establish a stronger result for the restricted so-called equal degree factoring problem. The restricted problem assumes that all irreducible factors have the same known degree, in which case we can factor an n degree polynomial in $O(n^{2.69+1.69a})$ expected fixed precision operations. Furthermore, one may test a polynomial for irreducibility, or if all its irreducible factors have the same degree, in $O(n^{2.69+1.69a})$ deterministic fixed precision integer operations. Below we will also give the precise complexity for general p and k. Our solution is a twist of an algorithm by von zur Gathen and Shoup [10]. As an intermediate result, we solve the problem of computing x^q modulo a polynomial of degree n over $\mathbb{F}_q$ faster than $\Theta(n(\log q)^2)$ fixed precision operations $= \Theta(n)$ (polynomial arithmetic) $\times \log q$ (raising to power of q) $\times\ \Theta(\log q)$ (fixed precision cost for arithmetic in $\mathbb{F}_q$).

---

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-9319776 (Erich Kaltofen). Part of the work was done while Victor Shoup was at Bellcore in Morristown, New Jersey, USA.

†Note that $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$, i.e., $g(n)$ is an asymptotic lower bound for $f(n)$. If in addition $f(n) = O(g(n))$ we write $f(n) = \Theta(g(n))$, i.e., $f(n)$ and $g(n)$ have the same asymptotic behavior.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. ISSAC'97, Maui, Hawaii, USA. ©1997 ACM 0-89791-875-4/97/ 0007 $ 3.50

2 Known Facts

The distinct degree factorization algorithm separates those factors that have differing degrees. It is based on a basic fact about finite fields,

$$x^{q^i} - x = \prod_{\substack{f \text{ irreducible over } \mathbb{F}_q \\ \deg(f) \text{ divides } i}} f(x).$$

This factorization implies several very useful algorithms. For one, it is easy to test f for irreducibility. One must have, with $n = \deg(f)$, that $x^{q^n} - x \equiv 0 \pmod{f(x)}$ and $\mathrm{GCD}(f(x), x^{q^{n/t}} - x) = 1$ for all prime numbers t dividing n. This test can be improved [24, Section 6]. A similar criterion (see von zur Gathen and Shoup [10, Fact 7.4]) tests if all factors of f have the same degree and if so determines that common degree.

Next, we give the classical distinct degree factorization algorithm, which as Joachim von zur Gathen tells us was already known to Gauss; Lidl and Niederreiter [19] attribute the method to Arwin [1]. First, we write

$$f^{[i]} = \prod_{\substack{g \text{ irred. factor of } f \\ \deg(g) = i}} g$$

The algorithm finds all factors $f^{[i]}$ of f as follows.

$f^* \leftarrow f$; `/* squarefree */`
`for` $i \leftarrow 1, \dots, \lfloor n/2 \rfloor$ `do`
 `{`$f^{[i]}(x) \leftarrow \mathrm{GCD}(-x + x^{q^i} \bmod f^*(x), f^*(x))$;
  $f^* \leftarrow f^* / f^{[i]}$;
 `}`
$f^{[\deg(f^*)]} \leftarrow f^*$; `/* factor with degree >` $\lfloor n/2 \rfloor$`*/`

This algorithm requires fast polynomial residue arithmetic, fast polynomial greatest common divisor (GCD) computation, computing high modular powers of x, etc. Table 1 summarizes the major algorithms employed for these tasks. There $f(x) \in \mathbb{F}_q[x]$ has degree n, and $g(x)$, $h(x)$ are modular residues. All counts are in terms of arithmetic operations in $\mathbb{F}_q$. It is useful to express the running times in terms of $M(n)$, the function that bounds the asymptotic complexity of the used polynomial multiplication algorithm, and $C(n)$, the function that that bounds the used modular polynomial composition algorithm, which is problem 4 in Table 1. The best $M(n)$ is $n \log n \operatorname{loglog} n$.

We shall expand on the meaning of this entry 4 in Table 1. For fast modular polynomial composition matrix multiplication is used. By ω we shall denote an (achievable) matrix multiplication exponent, i.e., when there is an algorithm that multiplies two $n \times n$ matrices in $O(n^\omega)$ arithmetic operations. For classical multiplication $\omega = 3$ and the fastest method has $\omega = 2.375477$ [9]. The following is an adaptation of a result by Brent and Kung [5] (see [10, fact 5.1]).

Lemma 1 *Let F be a monic polynomial in $R[x]$ of degree n, where R is a commutative ring, and let G and H be polynomials in $R[x]$ of degree $< n$. Then the polynomial composition of G and H modulo F, $G(H(x)) \bmod F(x)$ can be computed in $O(n^{(\omega+1)/2})$ arithmetic operations in R. In particular, the running time $O(n^{1.69})$ can be achieved.*

On January 15, 1997 we received an electronic message from Xiaohan Huang and Victor Pan that the matrix multiplication problem of dimensions $\lfloor\sqrt{n}\rfloor \times \lfloor\sqrt{n}\rfloor$ times $\lfloor\sqrt{n}\rfloor \times n$ used for the proof of Lemma 1 can be performed in $O(n^{1.666977})$ arithmetic operations [13]. One immediately obtains slight improvements of the estimates in Lemma 1 to $O(n^{1.67})$ and of all expontents given in the following sections (replace the decimal digits .69 by .67). Indeed, many of the new factorizations algorithms [10, 15] are speeded by improving the running time of the modular polynomial composition problem. In our theorems we express the running times in terms of $C(n)$, a function that bounds the asymptotic cost of modular polynomial composition. Using the algorithm by Brent and Kung we obtain $C(n) = n^{1.69}$ and using the one by Huang and Pan $C(n) = n^{1.67}$.

3 The Equal Degree Factorization Algorithm

The assumption for our factoring problem over $\mathbb{F}_q$, where $q = p^k$ and p a prime number, is that the input polynomial f has irreducible factors of one and the same degree d, which is known. The polynomial is free of repeated factors.

Algorithm E This algorithm takes as input a square-free polynomial $f \in \mathbb{F}_q[x]$ of degree n and a positive integer d. The input polynomial must be a product of $r = n/d$ irreducible polynomials of degree d. The output is the list $f_1, \dots, f_r$ of the irreducible factors of f.

Step E1 Pick a random element $\alpha \in \mathbb{A}_f$, where $\mathbb{A}_f = \mathbb{F}_q[x]/(f(x))$ is the polynomial algebra isomorphic to $\mathbb{F}_{q^d}^r$.

In $\mathbb{A}_f$ compute the following trace-like map:

$$\beta = T_{\mathbb{F}_{q^d}/\mathbb{F}_p}(\alpha) = \alpha + \alpha^p + \alpha^{p^2} + \cdots + \alpha^{p^{kd-1}}. \quad (1)$$

Step E2 If $p > 2$, compute the following quadratic residuosity-like map: $\gamma = \beta^{(p-1)/2}$; if $p = 2$ set $\gamma = \beta$.

Step E3 Let $\gamma = (g \bmod f)$. Recursively factor $g_1 = \mathrm{GCD}(g, f)$, $g_2 = \mathrm{GCD}(1 + g, f)$ and, if $p > 2$, $g_3 = f/(g_1 g_2)$.

The possibility of splitting the equal degree factors, similarly to the distinct degree factorization algorithm of Section 2, appears to have been first realized by Cantor and Zassenhaus [8]. They use the map $\alpha^{(q^d-1)/2}$ if $p > 2$ and work in a quadratic extension of $\mathbb{F}_{q^d}$ if $p = 2$. Ben-Or [2] introduced the trace of the Frobenius map used in Step E1, which is also used in the algorithm of von zur Gathen and Shoup [10].

Algorithm E is based on the fact that the trace of Frobenius maps random elements of $\mathbb{F}_{q^d}$, the images of α modulo the irreducible factors of f, to random residues of $\mathbb{F}_p$. This is most easily seen from the identity

$$x^{q^d} - x = \prod_{a \in \mathbb{F}_p} (a + x + x^p + \cdots + x^{p^{kd-1}}).$$

Since $\mathbb{F}_{q^d}$ is the splitting field of the left side and any element of $\mathbb{F}_{q^d}$ is a root of the right side, the left side must divide the right side. As the degrees and leading coefficients agree, both sides must be equal. The expected depth of the recursion is $O(\log r)$ [10, proof of theorem 3.7], therefore the running time of a single invocation determines within a factor of $O(\log r)$ the overall expected running time.

Problem	Running time	In terms of $M(n)$, $C(n)$	Inventors of algorithm
1. $g \cdot h \pmod f$	$O(n(\log n) \operatorname{loglog} n)$	$O(M(n))$	Schönhage & Strassen 1969 [23] Schönhage 1977 [22] ($p = 2$) Sieveking 1972 [25]/Kung 1974 [18]
2. $\mathrm{GCD}(f, g)$	$O(n(\log n)^2 \operatorname{loglog} n)$	$O(M(n) \log n)$	Knuth 1971 [16]/Moenck 1973 [20]
3. $g^q \pmod f$	$O(n^{1+o(1)} \log q)$	$O(M(n) \log q)$	using Pingala 200 b.c. (see [17, Sec. 4.6.3])
4. $g(h(x)) \pmod{f(x)}$	$O(n^{1.69})$	$O(C(n))$	Brent & Kung 1978 [5] Coppersmith & Winograd 1987 [9]
5. $x^{q^n} \pmod{f(x)}$ given $x^q \pmod{f(x)}$	$O(n^{1.69})$	$O(C(n) \log n)$	von zur Gathen & Shoup 1991 [10]
6. $g(h_1), ..., g(h_n) \pmod f$	$O(n^{2+o(1)})$	$O(M(n^2) \log n)$	using Moenck & Borodin 1972 [4, 10]
7. $x^{q^2}, ..., x^{q^n} \pmod{f(x)}$ given $x^q \pmod{f(x)}$	$O(n^{2+o(1)})$	$O(M(n^2) \log n)$	von zur Gathen & Shoup 1991 [10]

Table 1: Arithmetic cost of basic polynomial arithmetic over $\mathbb{F}_q$

4 Fast Trace of Frobenius Maps

The running time of Algorithm E is largely dependent on the speed of computing the trace of the Frobenius map in $\mathbb{A}_f$. It is this step that we speed up. We make the following assumption about the representation of elements in $\mathbb{A}_f$. The coefficient field $\mathbb{F}_q$, where $q = p^k$, is represented as $\mathbb{F}_q = \mathbb{F}_p[z]/(\varphi(z))$ where φ is a monic irreducible polynomial over $\mathbb{F}_p$ of degree k. The input polynomial $f(x) \in \mathbb{F}_q[x]$ is monic and square-free. Elements in the algebra $\mathbb{A}_f = \mathbb{F}_p[x, z]/(f(x, z), \varphi(z))$ are represented as polynomials in x and z with degree in x less than n and degree in z less than k.

Algorithm T This algorithm takes as input a polynomial $f \in \mathbb{F}_q[x]$, an element α in the algebra $\mathbb{A}_f = \mathbb{F}_q[x]/(f(x))$, and an integer $i > 1$.

It returns the quantities

$$\beta_i = \alpha^p + \alpha^{p^2} + \cdots + \alpha^{p^i} \in \mathbb{A}_f,$$
$$\xi_i = (x^{p^i} \bmod f(x)) \in \mathbb{A}_f,$$
$$\zeta_i = (z^{p^i} \bmod \varphi(z)) \in \mathbb{F}_q.$$

Step T1 Let $j = \lfloor i/2 \rfloor$. Recursively compute β_j, ξ_j, and ζ_j. Note that if $i = 1$ $\beta_1 = \alpha^p$ is found by binary exponentiation, as is ξ_1 and ζ_1.

Step T2 Compute

$$\beta_j^{p^j} = (\alpha^p + \cdots + \alpha^{p^j})^{p^j} = \alpha^{p^{j+1}} + \cdots + \alpha^{p^{2j}}$$

as follows:

$$\begin{aligned} \beta_j^{p^j} &= \left(\sum_{l=0}^{n-1} c_l(z) x^l \right)^{p^j} \bmod (f(x), \varphi(z)) \\ &= \sum_{l=0}^{n-1} c_l(z)^{p^j} (x^l)^{p^j} \bmod (f(x), \varphi(z)) \\ &= \sum_{l=0}^{n-1} c_l(z^{p^j})(x^{p^j})^l \bmod (f(x), \varphi(z)) \\ &= \sum_{l=0}^{n-1} c_l(\zeta_j)\, \xi_j^l \bmod (f(x), \varphi(z)); \end{aligned}$$

thus first compute for $l = 0, \ldots, n-1$, $\tilde{c}_l(z) = (c_l(\zeta_j) \bmod \varphi(z))$ by modular polynomial composition over $\mathbb{F}_p$. Then compute $\beta_j^{p^j} = \sum_{l=0}^{n-1} \tilde{c}_l(z) \xi_j^l \bmod (f(x), \varphi(z))$ by modular polynomial composition over $\mathbb{F}_q$. Finally, set $\beta_{2j} = \beta_j + \beta_j^{p^j}$.

Similarly, compute $\xi_{2j} = \xi_j^{p^j}$ and $\zeta_{2j} = \zeta_j^{p^j}$. If $i = 2j$ one is done. Otherwise, perform the next step.

Step T3 Compute $\beta_{2j+1} = \beta_1 + \beta_{2j}^p$, where $\beta_{2j}^p(x, z) = \beta_{2j}(\xi_1, \zeta_1) \bmod (f(x), \varphi(z))$. Similarly, compute $\xi_{2j+1} = \xi_{2j}^p$ and $\zeta_{2j+1} = \zeta_{2j}^p$.

We now estimate the running time of Algorithms T and E. Arithmetic in $\mathbb{F}_q$ is reduced to polynomial arithmetic over $\mathbb{F}_p$. The costliest arithmetic operation is the reciprocal, which can be accomplished in $O(M(k) \log k)$ operations in $\mathbb{F}_p$. Each recursive invocation of Algorithm T performs $O(n)$ modular polynomial compositions over $\mathbb{F}_p$ at a total cost of $O(nC(k))$ arithmetic steps in $\mathbb{F}_p$. Furthermore, $O(1)$ modular polynomial compositions over $\mathbb{F}_q$ are executed, at a total cost of $O(C(n)M(k))$ operations in $\mathbb{F}_p$. The remaining additions are dominated by these measures, except the computation of β_1 and ξ_1, which are done in $O(\log(p) M(n) M(k))$ operations in $\mathbb{F}_p$. The expected depth of the recursion tree of Algorithm E is $O(\log r)$. Each call on each level performs $O(\log(kd))$ invocations of Algorithm T, and one GCD computation over $\mathbb{F}_q$. Other than making f monic at the beginning, only in this GCD computation one performs O(input degree) reciprocals. Therefore, the extra $\log k$ factor introduced by the reciprocals in the GCDs is accounted for in the following estimates.

Theorem 1 *For $\mathbb{F}_q = \mathbb{F}_p[z]/(\varphi(z))$ with $\deg(\varphi) = k$ Algorithm E can be implemented so as to use*

$$O(\ \log(r)\ \log(kn)\ (n\ C(k) + C(n)\ M(k) + \log(p)\ M(n)\ M(k))\ )$$

expected operations in $\mathbb{F}_p$.

Note that the $\log r$ factor can be removed from some of the summands in the above asymptotic estimate by the techniques of [10, Section 4]. In the special case where $q = 2^n$ the running time is with the fastest known matrix exponent $O(n^{2.69})$ bit operations, that is, sub-cubic.

As explained in Section 2, a fast algorithm for computing $x^{q^{n/t}} \bmod f(x)$ for all prime numbers t dividing n and a fast greatest common divisor algorithm results in an efficient irreducibility test. Similarly, [10, Fact 7.4] one can test if the irreducible factors have all one and the same degree and determine that degree. These tests do not require randomizations. Let $\varpi(n)$ be the number of distinct prime factors of n; e.g., $\varpi(3^2 5) = 2$. In the worst case, $\varpi(n) = O(\frac{\log n}{\log\log n})$, while in the average case $\varpi(n) = O(\log\log n)$ [11, Section 22.10]. Using the methods of Shoup [24, Section 6], one easily obtains the following theorem.

Theorem 2 *For $\mathbb{F}_q = \mathbb{F}_p[z]/(\varphi(z))$ with $\deg(\varphi) = k$ one can determine if a polynomial over $\mathbb{F}_q$ is irreducible, or if all its irreducible factors are of equal degree, and if so determine the common degree, with*

$$O(\ \log(\varpi(n))\ \log(kn)(n\ C(k) + C(n)\ M(k) + \log(p)\ M(n)\ M(k))\ )$$

deterministic operations in $\mathbb{F}_p$.

5 Distinct Degree Factorization

We can also apply the methods of Section 4 to the distinct degree factorization algorithm of von zur Gathen and Shoup [10], or the black box Berlekamp algorithm [14, 15], and speed the entire factorization process for high algebraic extension. For simplicity, we describe the changes to the distinct degree factorization algorithm of von zur Gathen and Shoup. The running time of their algorithm is

$$O(n^{2+o(1)} + n^{1+o(1)} \log q) \qquad (2)$$

expected operations in $\mathbb{F}_q$, where n is the degree of the input polynomial f. Their idea is based on the distinct degree algorithm given in Section 2 and computes from $x^q \bmod f(x)$ the powers

$$x^{q^2} \bmod f(x), \ldots, x^{q^n} \bmod f(x)$$

using $O(n^{2+o(1)})$ field operations (see Table 1, Problems 6 and 7). For $q = p^k$ with $k = O(n^{1+a})$, where $a > 0$ is constant, this leads under the Kronecker model for arithmetic in $\mathbb{F}_q$ to

$$O(n^{3+2a+o(1)} \log p)$$

expected operations in $\mathbb{F}_p$ for computing the full distinct degree factorization of f.

We now observe that the $n^{1+o(1)} \log q$ term in (2) is contributed solely from the computation of $x^q \bmod f(x)$ over $\mathbb{F}_q$ (and the cost of equal degree factorization). However, in Section 4 we give a method for computing $x^q \bmod f(x)$ in

$$O(n^{2.69+1.69a} + n^{2+a+o(1)} \log p)$$

arithmetic operations in $\mathbb{F}_p$. Here and in the following the summand 0.69 represents (the exponent in $C(n)$) $- 1$. Thus we have the following improvement to general factorization, relying on the Cantor/Zassenhaus approach with the von zur Gathen and Shoup distinct degree factorization algorithm and our equal degree modification.

Theorem 3 *A polynomial over $\mathbb{F}_q$ with $q = p^k$ can be factored into its irreducible factors in*

$$O(\ M(n^2)\ \log(n)\ M(k) + \log(n)\ \log(nk)\ \ (n\ C(k) + \log(p)\ M(n)\ M(k))\ )$$

expected operations in $\mathbb{F}_p$. In particular, if $k = O(n^{1+a})$, where $a > 0$ is constant, a polynomial can be factored into its irreducible factors in

$$O(n^{3+a+o(1)} + n^{2.69+1.69a} + n^{2+a+o(1)} \log p) \qquad (3)$$

expected operations in $\mathbb{F}_p$.

The following observation may further clarify the result. Let $q = 2^k$ with $k = \Omega(n^{1.46})$. The von zur Gathen/Shoup factorization method, as well as the fast Berlekamp method, consumes $O(n(\log q)^{2+o(1)})$ fixed precision integer operations, while our speeded method only uses $O(n(\log q)^{1.69})$ fixed precision operations.

We wish to remark that the $n^{2.69+1.69a}$ term in the running time (3) of the above theorem, or an $n^{2.67+1.67a}$ term derived from a $C(k) = O(k^{1.67})$ modular polynomial composition algorithm (see last paragraph of Section 2), can be further improved if one carries out the n modular polynomial compositions of degree k needed in Step T2 of Algorithm T simultaneously by a single rectangular matrix multiplication [12].

Acknowledgements: We like to thank the two referees for catching several inaccuracies, and Xiaohan Huang for sending us his new results on fast rectangular matrix multiplication.

References

[1] Arwin, A. Über die Kongruenzen von dem fünften und höheren Graden nach einem Primzahlmodulus. *Arkiv f. matematik, astronom. o. fysik 14* (1918), 1–46. In German.

[2] Ben-Or, M. Probabilistic algorithms in finite fields. In *Proc. 22nd IEEE Symp. Foundations Comp. Sci.* (1981), pp. 394–398.

[3] Berlekamp, E. R. Factoring polynomials over large finite fields. *Math. Comp. 24* (1970), 713–735.

[4] Borodin, A., and Moenck, R. Fast modular transforms. *J. Comput. System Sci. 8* (1974), 366–386.

[5] Brent, R. P., and Kung, H. T. Fast algorithms for manipulating formal power series. *J. ACM 25*, 4 (1978), 581–595.

[6] Camion, P. Un algorithme de construction des idempotents primitifs d'ideaux d'algebres sur $\mathbb{F}_q$. *Ann. Discrete Math 12* (1982), 55–63.

[7] Cantor, D. G., and Kaltofen, E. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform. 28*, 7 (1991), 693–701.

[8] Cantor, D. G., and Zassenhaus, H. A new algorithm for factoring polynomials over finite fields. *Math. Comp. 36* (1981), 587–592.

[9] Coppersmith, D., and Winograd, S. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput. 9*, 3 (1990), 251–280.

[10] von zur Gathen, J., and Shoup, V. Computing Frobenius maps and factoring polynomials. *Comput. Complexity 2* (1992), 187–224.

[11] Hardy, G. H., and Wright, E. M. *An Introduction to the Theory of Numbers*, 5 ed. Oxford Univ. Press, Oxford, 1979.

[12] Huang, X. Private communication, Apr. 1997.

[13] Huang, X., and Pan, V. Fast rectangular matrix multiplications and improving parallel matrix computations. In *Proc. Second Internat. Symp. Parallel Symbolic Comput. PASCO '97* (New York, N. Y., 1997), M. Hitz and E. Kaltofen, Eds., ACM Press. To appear.

[14] Kaltofen, E., and Lobo, A. Factoring high-degree polynomials by the black box Berlekamp algorithm. In *Proc. Internat. Symp. Symbolic Algebraic Comput. ISSAC '94* (New York, N. Y., 1994), ACM Press, pp. 90–98.

[15] Kaltofen, E., and Shoup, V. Subquadratic-time factoring of polynomials over finite fields. In *Proc. 27th Annual ACM Symp. Theory Comp.* (New York, N.Y., 1995), ACM Press, pp. 398–406. *Math. Comput.*, in press.

[16] Knuth, D. E. The analysis of algorithms. *Actes du congrès international des Mathématiciens 3* (1970), 269–274.

[17] Knuth, D. E. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Ed. 2.* Addison Wesley, Reading, MA, 1981.

[18] Kung, H. T. On computing reciprocals of power series. *Numer. Math. 22* (1974), 341–348.

[19] Lidl, R., and Niederreiter, H. *Finite Fields.* Addison-Wesley, Reading, MA, 1983.

[20] Moenck, R. T. Fast computation of GCDs. *Proc. 5th ACM Symp. Theory Comp.* (1973), 142–151.

[21] Rabin, M. O. Probabilistic algorithms in finite fields. *SIAM J. Comp. 9* (1980), 273–280.

[22] Schönhage, A. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inform. 7* (1977), 395–398. In German.

[23] Schönhage, A., and Strassen, V. Schnelle Multiplikation grosser Zahlen. *Computing 7* (1971), 281–292. In German.

[24] Shoup, V. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput. 17*, 5 (May 1994), 371–391.

[25] Sieveking, M. An algorithm for division of power series. *Computing 10* (1972), 153–156.