A Fast Deterministic Algorithm for Factoring Polynomials over Finite Fields of Small Characteristic

Victor Shoup
Computer Sciences Department
University of Toronto
Toronto, Ontario M5S 1A4

Abstract

We present a new algorithm for factoring polynomials over finite fields. Our algorithm is deterministic, and its running time is "almost" quadratic when the characteristic is a small fixed prime. As such, our algorithm is asymptotically faster than previously known deterministic algorithms for factoring polynomials over finite fields of small characteristic.

Appeared in *Proc. 1991 International Symposium on Symbolic and Algebraic Computation* **(ISSAC)**, pp. 14–21, 1991.

1. Introduction

Consider the problem of factoring a univariate polynomial f of degree n over the finite field $\mathbf{F}_q$, where $q = p^k$ and p is a small, fixed prime. We assume that $\mathbf{F}_q$ is represented as $\mathbf{F}_p(\theta)$, where θ is the root of an irreducible polynomial over $\mathbf{F}_p$ of degree k. We present a new deterministic algorithm for this problem whose asymptotic complexity is less than that of previous deterministic algorithms.

In discussing running times of algorithms, for expositional purposes we treat p as a constant in Sections 1 and 2 of this paper; however, in Section 4 we will make explicit the dependence of our algorithm on p. Furthermore, we will use the "Soft-O" notation to suppress logarithmic factors in running time estimates: we say that $g = \tilde{O}(h)$ iff for some constant c, $g = O(h(\log h)^c)$.

Besides the general factoring problem, we consider as a special case the *equal degree factoring problem*, in which the input is a polynomial over $\mathbf{F}_q$ that is the product of m distinct monic irreducible factors, each of the same degree d. Setting $d = 1$, this includes as a special case the *root finding problem.*

We can now state our new results more precisely:

1. We can solve the general factoring problem with a deterministic algorithm whose running time is $\tilde{O}((nk)^2)$.

2. We can solve the equal degree factoring problem with a deterministic algorithm whose running time is $\tilde{O}(m(dk)^2)$.

We briefly compare our algorithm to other known algorithms. In the next section, this comparison is done in greater detail.

In the case where $k = 1$, and we are factoring over the prime field $\mathbf{F}_p$, the running time $\tilde{O}(n^2)$ was previously obtained by the algorithm of the present author in [13]. The method described in that paper does not appear to generalize to large extensions of $\mathbf{F}_p$.

Using linear algebra techniques, the general factoring problem can be solved in time $O((nk)^\omega)$, where ω is the exponent of matrix multiplication (see [15]). Currently, the best value for ω is $\omega \approx 2.376$ [10]. In order to achieve our running time bounds, our algorithm avoids linear algebra.

Our algorithm is actually a generalization of Berlekamp's trace algorithm for root finding [3]. This algorithm involves the computation of many trace functions; if computed separately, the cost of computing these trace functions would be too much. To achieve our running time bounds, we employ a new technique for computing several trace functions in not much more time than that required to compute just one trace function.

Finally, if probabilistic algorithms are allowed, then the running time bounds stated in Results 1 and 2 above have already been obtained with the algorithms of Ben-Or [2] and Cantor/Zassenhaus [9]. The significance of our results is that our algorithm is deterministic.

2. Overview

In this section, we outline the main ingredients of our algorithm, along the way comparing our algorithm with other known algorithms. We begin by summarizing some well-known facts about the complexity of various arithmetic operations.

By a ring R we shall always mean a commutative ring with unity, and by an R-operation, we mean addition, subtraction, or multiplication of two elements of R. For a field F, by an F-operation, we mean addition, subtraction, multiplication, or division of two elements of F. We let $M(t)$ denote the number of R-operations required to compute the product of two degree t polynomials in $R[X]$.

It is shown in [8] that $M(t) = O(t \log t \log\log t)$. We quote the following well-known results; the proofs can be found, e.g., in [4].

Theorem 2.1. *Let R be a ring, and let F be a field.*

(1) *Let f be a monic polynomial in $R[X]$ of degree $\leq t$. Then with $O(M(t))$ R-operations we can compute a degree t approximation to the multiplicative inverse of f in the ring of formal power series over R.*

(2) *Let f and g be polynomials in $R[X]$ of degree $\leq t$ and assume that g is monic. Then $f \bmod g$ can be computed using $O(M(t))$ R-operations.*

(3) *Let $f, g_1, \ldots, g_k$ be polynomials in $R[X]$ such that $\deg f \leq t$, $\deg g_1 + \cdots + \deg g_k \leq t$, and the g_i's are monic. Then $f \bmod g_1, \ldots, f \bmod g_k$ can be computed using $O(M(t)\log k)$ R-operations.*

(4) *Let f and g be polynomials in $F[X]$ of degree $\leq t$. Then the greatest common divisor d of f and g can be computed using $O(M(t)\log t)$ F-operations. Moreover, polynomials $a, b \in F[X]$ of degree $O(t)$ satisfying $af + bg = d$ can be computed in the same time bound.*

(5) *Let $\alpha \in R$. Then for any integer $m > 0$, α^m can be computed using $O(\log m)$ multiplications in R.*

We can quickly reduce the general factoring problem to the equal degree factoring problem by a process called distinct degree factorization. Given a polynomial f of degree n, we construct polynomials $f^{(1)}, \ldots, f^{(n)}$ such that $f^{(i)}$ is the product of all the distinct monic irreducible polynomials over $\mathbf{F}_q$ of degree i that divide f. This is accomplished by using the fact that the polynomial $X^{q^i} - X$ is the product of all distinct monic irreducible polynomials whose degree divides i. We refer the reader to [12] for details, and note that using fast algorithms for polynomial arithmetic, we can perform the distinct degree factorization process in time $\tilde{O}((nk)^2)$.

We therefore assume that the polynomial f that we wish to factor is the product of m distinct monic irreducible polynomials, each of degree d:

$$f = f_1 \cdots f_m.$$

The degree of f is $n = md$.

Consider the $\mathbf{F}_q$-algebra $R = \mathbf{F}_q[X]/(f)$. Let $\lambda = (X \bmod f)$ be the image X in R. By the complexity results mentioned above, each $\mathbf{F}_q$-operation can be performed in time $\tilde{O}(k)$, and each R-operation can be performed in time $\tilde{O}(mdk)$.

By the Chinese Remainder Theorem, for polynomials $g \in \mathbf{F}_q[X]$ the map defined by

$$(g \bmod f) \mapsto (g \bmod f_1, \ldots, g \bmod f_m)$$

is an $\mathbf{F}_q$-algebra isomorphism of R and the direct sum

$$\bigoplus_{i=1}^{m} \mathbf{F}_q[X]/(f_i) \cong \bigoplus_{i=1}^{m} \mathbf{F}_{q^d}.$$

This isomorphism maps $\mathbf{F}_q$ onto the diagonal, i.e. for $a \in \mathbf{F}_q$, $a \mapsto (a, \ldots, a)$. For $1 \leq i \leq m$ let $\alpha \mapsto \alpha^{(i)}$ be the projection map of $\alpha \in R$ onto the i-th summand $R^{(i)} = \mathbf{F}_q[X]/(f_i)$.

We now isolate two well-known subalgebras of R. The *relative Berlekamp subalgebra* B consists of all elements $\alpha \in R$ such that $\alpha^{(i)} \in \mathbf{F}_q$ for all $1 \leq i \leq m$. The *Berlekamp subalgebra* A consists of all elements $\alpha \in R$ such that $\alpha^{(i)} \in \mathbf{F}_p$ for all $1 \le i \le m$. The following inclusion diagram helps to describe the situation.

$$\begin{tikzcd}
& R \arrow[d, dash] & \\
& B \arrow[dl, dash] \arrow[dr, dash] & \\
\mathbf{F}_q \arrow[dr, dash] & & A \arrow[dl, dash] \\
& \mathbf{F}_p &
\end{tikzcd}$$

A set $S \subset B$ is called a *relative separating set* if for all $1 \le i, j \le m$ with $i \ne j$, there exists an element $\alpha \in S$ such that $\alpha^{(i)} \ne \alpha^{(j)}$. If S satisfies the further condition that $S \subset A$, then S is simply called a *separating set.*

Separating sets are very useful in factoring polynomials if p is small. Consider two distinct irreducible factors f_i and f_j of f. If S is a separating set, then we know that for some $\alpha \in S$, $\alpha^{(i)} \ne \alpha^{(j)}$. Say $\alpha^{(i)} = a \in \mathbf{F}_p$. Then if $\alpha = (g \bmod f)$, where $g \in \mathbf{F}_q[X]$, then $g - a$ is divisible by f_i but not by f_j. Therefore, given g, if we consider $\gcd(g - \delta, f)$ for all $\delta \in \mathbf{F}_p$, we are guaranteed to split f into two factors, one divisible by f_i, and the other by f_j. If p is small (as we are assuming) and the size of S is not too large, we can use this idea to obtain an efficient algorithm to *completely* factor f.

The computation of a separating set turns out to be the bottleneck in deterministic factoring algorithms, so we consider some of the known methods for this computation.

The oldest method goes back to Berlekamp [3]. Any $\mathbf{F}_p$-basis for A is clearly a separating set. Furthermore, A is easily seen to be the kernel of the $\mathbf{F}_p$-linear map $\alpha \mapsto \alpha^p - \alpha$. We can compute the matrix of this linear transformation with respect to the basis $\{\lambda^\mu \theta^\nu : 0 \le \mu < n, 0 \le \nu < k\}$, and then by diagonalizing this matrix we can obtain a basis for A. The total time for this computation is dominated by the time required to diagonalize a $kn \times kn$ matrix over $\mathbf{F}_p$, which is $O((nk)^\omega)$.

Our approach to computing a separating set begins by computing a relative separating set.

One method for computing a relative separating set, which is described by Camion [6], runs as follows. Let $T_{R/B}$ be the map that sends $\alpha \in R$ to $\alpha + \alpha^q + \cdots + \alpha^{q^{d-1}}$. So $T_{R/B}$ acts on $R^{(i)}$ as the trace from $\mathbf{F}_{q^d}$ down to $\mathbf{F}_q$ (and hence is a $\mathbf{F}_q$-linear map from R onto B). In [6], it is shown that the set $\{T_{R/B}(\lambda^\mu) : 0 \le \mu < 2d\}$ is a relative separating set.

One application of $T_{R/B}$ requires $O(dk)$ R-operations. The obvious method for computing the elements in this relative separating set requires $2d$ such applications, and hence takes time $\tilde{O}(md^3k^2)$.

Our method for constructing a relative separating set, described in [13], is the following. Consider the ring $R[Y]$ of univariate polynomials over R. Let $h = (Y - \lambda)(Y - \lambda^q) \cdots (Y - \lambda^{q^{d-1}}) \in R[Y]$, and write $h = h_0 + h_1 Y + \cdots h_{d-1} Y^{d-1} + Y^d$. In [13] it is shown that the set $S_0 = \{h_i : 0 \le i < d\}$ is a relative separating set; this fact follows directly from the observation that $h_i^{(j)}$ is the coefficient of X^i in f_j.

We can compute the powers $\lambda, \lambda^q, \ldots, \lambda^{q^{d-1}}$ with $O(dk)$ R-operations, and then using a fast algorithm for multiplying polynomials in $R[Y]$, we can compute the coefficients of h with $\tilde{O}(d)$ R-operations. Thus, we can compute the elements of S_0 in time $\tilde{O}(m(dk)^2)$.

Generalizing the idea of Berlekamp's trace algorithm for root finding [3], we can use a relative separating set S to construct a separating set S' as follows. Let $T_{B/A}$ be the map that sends $\alpha \in R$ to $\alpha + \alpha^p + \cdots + \alpha^{p^{k-1}}$. $T_{B/A}$ acts on $B^{(i)}$ as the trace from $\mathbf{F}_q$ down to $\mathbf{F}_p$ (and hence is a $\mathbf{F}_p$-linear map that maps B onto A). It can easily be shown that if S is a relative separating set, then $S' = \{T_{B/A}(\theta^\nu \alpha) : 0 \leq \nu < k, \alpha \in S\}$ is a separating set; this is a direct consequence of the fact that for any pair of distinct elements $a, b \in \mathbf{F}_q$, there exists ν with $0 \leq \nu < k$ such that the trace from $\mathbf{F}_q$ to $\mathbf{F}_p$ maps $\theta^\nu a$ and $\theta^\nu b$ onto distinct elements in $\mathbf{F}_p$.

We shall apply this construction to the relative separating set S_0 described above to obtain a separating set S_1. An individual application of $T_{B/A}$ requires $O(k)$ R-operations, and hence takes time $\tilde{O}(mdk^2)$. The obvious method for computing S_1 requires dk such applications, and hence takes time $\tilde{O}(md^2k^3)$.

We are now ready to describe the new idea in our algorithm. Suppose we have elements $\alpha_0, \ldots, \alpha_{t-1}$ and $\beta_0, \ldots, \beta_{t-1}$ in some ring R. In the next section we show (Theorem 3.4) that with $\tilde{O}(t)$ R-operations we can compute all of the generalized power sums

$$\gamma_s = \sum_{i=0}^{t-1} \alpha_i^s \beta_i \qquad (s = 0, \ldots, t-1).$$

Now consider the problem of computing, for fixed $\alpha \in S_0$, the quantities $T_{B/A}(\theta^\nu \alpha)$ for $0 \leq \nu < k$. For $0 \leq i < k$, let $\theta_i = \theta^{p^i}$ and $\alpha_i = \alpha^{p^i}$. Then $T_{B/A}(\theta^\nu \alpha)$ is just the generalized power sum $\sum_{i=0}^{k-1} \theta_i^\nu \alpha_i$. We can clearly compute the θ_i's and α_i's using $O(k)$ R-operations, and then using Theorem 3.4 we can compute the the generalized power sums using $\tilde{O}(k)$ R-operations. Repeating this for each $\alpha \in S_0$, we obtain S_1 with $\tilde{O}(dk)$ R-operations, and hence in time $\tilde{O}(m(dk)^2)$.

Thus, the total time to compute the separating set S_1 using our new approach is $\tilde{O}(m(dk)^2)$.

We close this section with a brief description of how the *probabilistic* algorithms of Ben-Or and Cantor/Zassenhaus achieve a $\tilde{O}(m(dk)^2)$ *expected* running time. Consider the map $T_{R/A}$ which sends $\alpha \in R$ to $\alpha + \alpha^p + \cdots + \alpha^{p^{dk-1}}$. One can easily show that if we choose $\alpha_1, \alpha_2, \cdots$ at random from R, then the expected value of the least t such that the set $\{T_{R/A}(\alpha_i) : 1 \leq i \leq t\}$ is a separating set is $O(\log m)$. Since the time required to compute $T_{R/A}$ once is $\tilde{O}(m(dk)^2)$, this leads to an algorithm whose expected running time is $\tilde{O}(m(dk)^2)$.

3. Computing generalized power sums

In this section, we consider the following problem: given $\alpha_0, \ldots, \alpha_{t-1}$ and $\beta_0, \ldots, \beta_{t-1}$ in a ring R, compute the generalized power sums

$$\gamma_s = \sum_{i=0}^{t-1} \alpha_i^s \beta_i \qquad (s = 0, \ldots, t-1).$$

We shall present an algorithm for this problem that requires $O(M(t) \log t)$ R-operations, and space for $O(t)$ elements in R. There are a couple of other methods in the literature for solving this problem, which we discuss at the end of this section.

We will first need an algorithm for extending a linear recurrence sequence.

Theorem 3.1. *Let R be a commutative ring with unity. Suppose we are given elements $c_1, \ldots, c_t$ and $\gamma_0, \ldots, \gamma_{t-1}$ in R. For $s \geq t$ let γ_s be defined by the recurrence*

$$\gamma_s + c_1 \gamma_{s-1} + \cdots + c_t \gamma_{s-t} = 0. \tag{3.2}$$

We can compute $\gamma_t, \ldots, \gamma_{2t-1}$ using $O(M(t))$ R-operations.

Proof. Let $c_0 = 1$. We define the following polynomials.

$$\begin{aligned} G &= \sum_{j=0}^{t} c_j X^j \\ U &= \sum_{j=0}^{t-1} \gamma_j X^j \\ V &= \sum_{j=0}^{t-1} \gamma_{t+j} X^j \end{aligned}$$

We then consider the products

$$GU = \sum_{j=0}^{2t-1} u_j X^j$$

$$GV = \sum_{j=0}^{2t-1} v_j X^j$$

Calculating the coefficients of these products, and using (3.2), one finds that

$$u_{t+j} + v_j = 0 \qquad (j = 0, \ldots, t-1). \tag{3.3}$$

We can rewrite this as follows. Let $GU = F_0 + X^t F_1$, where F_0 and F_1 are polynomials of degree at most $t-1$. Then we can rewrite (3.3) as

$$GV \equiv -F_1 \pmod{X^t}.$$

Let H be the multiplicative inverse of G in the ring of formal power series over R. Then we have

$$V \equiv -HF_1 \pmod{X^t}.$$

Thus, to compute the coefficients of V (which are the quantities we desire), we do the following:

1. Compute a degree t polynomial approximation H^* of H.

2. Compute F_1 by computing the product GU, and throwing away the low order t terms.

3. Compute $-H^* F_1$, and throw away the high order n terms.

Since each step of this algorithm uses $O(M(t))$ R-operations, so does the entire algorithm. □

We now come to our algorithm for computing generalized power sums.

Theorem 3.4. *Let R be a commutative ring with unity. Suppose we are given elements $\alpha_0, \ldots, \alpha_{t-1}$ and $\beta_0, \ldots, \beta_{t-1}$ in R. For $s \geq 0$ let*

$$\gamma_s = \sum_{i=0}^{t-1} \alpha_i^s \beta_i.$$

We can compute $\gamma_0, \ldots, \gamma_{t-1}$ using $O(M(t) \log t)$ R-operations.

Proof. We assume that t is a power of 2 (otherwise, pad with zeros). Consider the polynomial

$$G = (1 - \alpha_0 X) \cdots (1 - \alpha_{t-1} X).$$

Let

$$G = \sum_{j=0}^{t} c_j X^j$$

and let

$$\tilde{G} = \sum_{j=0}^{t} c_{t-j} X^j = (X - \alpha_0) \cdots (X - \alpha_{t-1}).$$

Claim. For $s \geq t$, we have

$$\gamma_s + c_1 \gamma_{s-1} + \cdots + c_t \gamma_{s-t} = 0.$$

This claim can be seen as follows.

$$\begin{aligned} \sum_{j=0}^{t} c_j \gamma_{s-j} &= \sum_{j=0}^{t} c_j \sum_{i=0}^{t-1} \alpha_i^{s-j} \beta_i \qquad \text{(by definition)} \\ &= \sum_{i=0}^{t-1} \alpha_i^{s-t} \beta_i \sum_{j=0}^{t} c_j \alpha_i^{t-j} \\ &= \sum_{i=0}^{t-1} \alpha_i^{s-t} \beta_i \tilde{G}(\alpha_i) \\ &= 0 \qquad \text{(since each } \alpha_i \text{ is a zero of } \tilde{G}) \end{aligned}$$

With this claim and Theorem 3.1, we see that given $\gamma_0, \ldots, \gamma_{t-1}$, and given the coefficients of G, we can compute $\gamma_t, \ldots, \gamma_{2t-1}$ using $O(M(t))$ R-operations.

Consider the following recursive algorithm to compute generalized power sums. For convenience, the role of t is now played by $2t$. The input is $\alpha_0, \ldots, \alpha_{2t-1}$ and $\beta_0, \ldots, \beta_{2t-1}$. The output is

$$\gamma_s = \sum_{i=0}^{2t-1} \alpha_i^s \beta_i \qquad (s = 0, \ldots, 2t-1)$$

and the coefficients of the polynomial

$$G = (1 - \alpha_0 X) \cdots (1 - \alpha_{2t-1} X).$$

1. Divide the problem into two equal sized pieces, and recursively compute the following quantities:
   - $\gamma'_s = \sum_{i=0}^{t-1} \alpha_i^s \beta_i \qquad (s = 0, \ldots, t-1)$
   - the coefficients of $G' = (1 - \alpha_0 X) \cdots (1 - \alpha_{t-1} X)$
   - $\gamma''_s = \sum_{i=t}^{2t-1} \alpha_i^s \beta_i \qquad (s = 0, \ldots, t-1)$
   - the coefficients of $G'' = (1 - \alpha_t X) \cdots (1 - \alpha_{2t-1} X)$

2. Extend the length t sequences γ'_s and γ''_s to the corresponding sequences of length $2t$ using the coefficients of G' and G'', and the algorithm of Theorem 3.1.

3. For $s = 0, \ldots, 2t-1$, set $\gamma_s = \gamma'_s + \gamma''_s$, and compute the coefficients of $G = G'G''$.

The number of R-operations performed by this algorithm is determined by the recurrence

$$\begin{aligned} T(2t) &= 2T(t) + O(M(t)) \\ T(1) &= O(1), \end{aligned}$$

which has the solution

$$T(t) = O(M(t)\log t).$$

□

We can rephrase the problem of computing generalized power sums by saying that we want to compute the matrix-vector product $V^T x$, where V is the Vandermonde matrix $V = (\alpha_{i-1}^{j-1})$ and x is the column vector $x = (\beta_{i-1})$.

We mention two other methods for solving this problem. One is described in [7], where it is assumed that R is a field and that the α_i's are distinct. It has the same time and space complexity as ours. This algorithm works by first computing $y = V^{-1}x$, and then applying the Hankel matrix $V^T V$ to y. The use of this algorithm over an arbitrary ring R is hindered by the fact that it performs several divisions by elements in R, and it is not clear that these can easily be avoided.

There are general methods by which one can transform an algorithm for computing the matrix-vector product Ax into an algorithm with the same asymptotic running time for computing $A^T x$ [1, 11]. Since computing Vx is known to take $O(M(t)\log t)$ R-operations, the same bound applies to computing $V^T x$. The algorithms that result from these general transformations are quite "unnatural," and moreover, require space proportional to their running times.

Either of these methods would have sufficed in proving the main result of this paper; however, our algorithm is still perhaps of interest in itself, since it avoids divisions, requires space for only $O(t)$ R-elements, and has a fairly simple and natural description.

4. Algorithmic Details

In this section we supply the remaining details of our factoring method. In analyzing the running time, we make explicit the dependence on p. Running times are measured in terms of $\mathbf{F}_p$-operations.

Our factoring algorithm proceeds as follows.

First, we reduce the general factoring problem to the equal degree factoring problem by first performing distinct degree factorization. This can be done with $\log p \cdot \tilde{O}((nk)^2)$ $\mathbf{F}_p$-operations (see [12]). We therefore assume that f is the product of m distinct monic irreducible polynomials of degree d. All of the notation and terminology introduced in Section 2 is now in force.

Second, we compute the separating set S_1 described in Section 2. Using the methods described there, this takes $\log p \cdot \tilde{O}(m(dk)^2)$ $\mathbf{F}_p$-operations.

Third, we apply the following factorization procedure that takes as input the polynomial f and a corresponding separating set S. The output is the set of irreducible factors of f. We construct finer and finer partial factorizations $U \subset \mathbf{F}_q[X]$ consisting of monic polynomials with $\prod_{u \in U} u = f$. Initially, $U = \{f\}$.

```
while |U| < m do
        Choose s ∈ S, and then remove s from S
        δ ← 0
        while Test(U, s) do
                Refine(U, s + δ)
                if p ≠ 2 then Refine(U, (s + δ)^((p−1)/2) − 1)
                δ ← δ + 1
```

The factorization procedure makes use of two subroutines. The first is the operation $Refine(U, v)$, which, given a partial factorization U and a polynomial $v \in \mathbf{F}_q[X]$, replaces U by the refinement U' obtained in the following fashion: for each $u \in U$, if $\gcd(u, v)$ is a trivial divisor of u, put u in U'; otherwise, put $\gcd(u, v)$ and $u/\gcd(u, v)$ in U'. For polynomials v of degree less than n, we can perform the $Refine$ operation using $\tilde{O}(mdk)$ $\mathbf{F}_p$-operations. The second operation is $Test(U, v)$, which returns *true* if $(v \bmod u) \notin \mathbf{F}_p$ for some $u \in U$, and *false* otherwise. The value of $Test(U, v)$ indicates whether v is of any use in obtaining a further refinement of U. This can also be computed using $\tilde{O}(mdk)$ $\mathbf{F}_p$-operations.

To analyze the running time of this algorithm, for an odd prime p we define the quantity $B(p)$ as follows: let χ be the quadratic character on $\mathbf{F}_p$, and as a and b range over all pairs of distinct elements in $\mathbf{F}_p$, let $B(p)$ be the maximum value of B such that $\chi(a + \delta) = \chi(b + \delta)$ for $0 \leq \delta < B$. For $p = 2$, define $B(p) = 1$.

Since we are using the separating set S_1, which contains dk elements, the number of $\mathbf{F}_p$-operations used by our factorization procedure is easily seen to be bounded by

$$\log p \cdot \tilde{O}(m(dk)^2) + B(p) \log p \cdot \tilde{O}(mdk \cdot \min(m, dk)).$$

In [13] it was shown that $B(p) = O(p^{1/2} \log p)$. It was subsequently shown in [14] that $B(p) = O(p^{1/2})$.

Putting all of this together, we can state our results as follows:

Theorem 4.1.

1. *We can solve the general factoring problem using*

$$\log p \cdot \tilde{O}((nk)^2) + p^{1/2} \log p \cdot \tilde{O}((nk)^{3/2})$$

   $\mathbf{F}_p$*-operations.*

2. *We can solve the equal degree factoring problem using*

$$\log p \cdot \tilde{O}(m(dk)^2) + p^{1/2} \log p \cdot \tilde{O}(mdk \cdot \min(m, dk))$$

   $\mathbf{F}_p$*-operations.*

Remark. Several authors have fallaciously drawn the inference that the fact that the maximum number of consecutive quadratic residues or nonresidues mod p is $O(p^{1/4} \log p)$ (see [5]) implies that polynomials over a finite field of characteristic p can be factored in time proportional to $p^{1/4}$ times a polynomial in the input size. The relevant quantity is not the number of consecutive quadratic residues or nonresidues, but rather the quantity $B(p)$ defined above; the author is not aware of any bounds on $B(p)$ better that $O(p^{1/2})$. Improving this bound is an important open problem.

Acknowledgements

The author would like to thank Alan Borodin, Mark Giesbrecht, Erich Kaltofen, and Roman Smolensky for valuable discussions.

References

[1] W. Baur and V. Strassen. The complexity of computing partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

[2] M. Ben-Or. Probabilistic algorithms in finite fields. In *22nd Annual Symposium on Foundations of Computer Science*, pages 394–398, 1981.

[3] E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24(111):713–735, 1970.

[4] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numeric Problems.* American Elsevier, 1975.

[5] D. A. Burgess. A note on the distribution of residues and non-residues. *Jour. London Math. Soc.*, 38:253–256, 1963.

[6] P. Camion. Improving an algorithm for factoring polynomials over a finite field and constructing large irreducible polynomials. *IEEE Trans. Inform. Theory*, IT-29(3):378– 385, 1983.

[7] J. F. Canny, E. Kaltofen, and L. Yagati. Solving systems of non-linear polynomial equations faster. In *Proc. ACM-SIGSAM Int. Symp. on Symbolic and Algebraic Computation*, pages 121–128, 1989.

[8] D. G. Cantor and E. Kaltofen. Fast multiplication of polynomials over arbitrary rings. Technical Report 87-35, Department of Computer Science, Rensselaer Polytechnic Institute, 1987. To appear, *Acta. Inf.*

[9] D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Math. Comp.*, 36(154):587–592, 1981.

[10] D. Coppersmith and S. Winograd. Matrix multiplication via Behrend's method. In *19th Annual ACM Symposium on Theory of Computing*, pages 1–6, 1987.

[11] M. Kaminski, D. G. Kirkpatrick, and N. H. Bshouty. Addition requirements for matrix and transposed matrix products. *Journal of Algorithms*, 9:354–364, 1988.

[12] R. T. Moenck. On the efficiency of algorithms for polynomial factoring. *Math. Comp.*, 31(137):235–250, 1977.

[13] V. Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inform. Process. Lett.*, 33(5):261–267, 1990.

[14] I. E. Shparlinsky. On some questions in the theory of finite fields. Preprint, 1990.

[15] J. von zur Gathen. Factoring polynomials and primitive elements for special primes. *Theoret. Comput. Sci.*, 52:77–89, 1987.