THE BLACK-BOX NIEDERREITER ALGORITHM
AND ITS IMPLEMENTATION OVER THE BINARY FIELD

PETER FLEISCHMANN, MARKUS CHR. HOLDER, AND PETER ROELSE

Abstract. The most time-consuming part of the Niederreiter algorithm for factoring univariate polynomials over finite fields is the computation of elements of the nullspace of a certain matrix. This paper describes the so-called "black-box" Niederreiter algorithm, in which these elements are found by using a method developed by Wiedemann. The main advantages over an approach based on Gaussian elimination are that the matrix does not have to be stored in memory and that the computational complexity of this approach is lower. The black-box Niederreiter algorithm for factoring polynomials over the binary field was implemented in the C programming language, and benchmarks for factoring high-degree polynomials over this field are presented. These benchmarks include timings for both a sequential implementation and a parallel implementation running on a small cluster of workstations. In addition, the Wan algorithm, which was recently introduced, is described, and connections between (implementation aspects of) Wan’s and Niederreiter’s algorithm are given.

1. Introduction

Factoring univariate polynomials over finite fields into irreducible factors is a basic problem in the area of symbolic computation. Two well-known techniques for solving this problem are the classical approach by Berlekamp [1] and the approach proposed by Niederreiter in [13]. In both these algorithms, elements of the nullspace of a certain matrix are computed, after which factors of the polynomial can be extracted by suitable gcd-computations. The Niederreiter algorithm is very well suited for factoring polynomials over small finite fields, especially \( \mathbb{F}_2 \), which is of particular interest for practical applications. For a survey of the Niederreiter algorithm, refer to [12] or [13].

To compute elements of the nullspace, which is the most time-consuming part of the algorithms in practice, two different approaches can be followed. The first one, the so-called “explicit” approach, consists of setting up and storing the matrix (densely) in memory, after which Gaussian elimination can be used to find a basis for the nullspace. The advantage of this approach is that it is well-suited for parallelization. A C implementation for factoring polynomials over \( \mathbb{F}_2 \) with Niederreiter’s algorithm on an IBM SP2 is described in [17]. The main disadvantage is the memory requirement for this approach when high-degree polynomials are to be factored.

Received by the editor July 23, 2001 and, in revised form, February 7, 2002.
2000 Mathematics Subject Classification. Primary 11-04, 11T06, 11Y16.
Key words and phrases. Finite fields, polynomial factorization, implicit linear algebra.

©2003 American Mathematical Society
The second technique for finding elements of the nullspace is the so-called "implicit" (or black-box) approach, and is based on Wiedemann’s algorithm [19]. For both Berlekamp’s and Niederreiter’s algorithm, this approach has a lower computational complexity than the “explicit” one, and is therefore well-suited for factoring high-degree polynomials. The main advantage over the “explicit” approach is that the matrix does not have to be stored in memory, enabling efficient implementations on a moderately sized personal computer or workstation. Its main disadvantage is that parallelization of the algorithm is not as straightforward as with Gaussian elimination. An implementation of Berlekamp’s algorithm using this ”implicit" technique is described in [8].

This paper describes the black-box Niederreiter algorithm, and is organized as follows. Section 2 contains a brief description of the Niederreiter algorithm and a closely related algorithm proposed recently by Wan [18]. In Section 3 an overview of the black-box approach is given, including the algorithm used to find elements of the nullspace of Niederreiter’s (or Wan’s) algorithm. Section 4 contains the benchmarks for an implementation of this algorithm in the C programming language. Run-times for a sequential implementation and for a parallel implementation running on a small cluster of PC’s are presented in this section.

2. Niederreiter’s Algorithm

Let \( \mathbb{F}_q \) be the finite field with \( q = p^s \) elements and \( f \in \mathbb{F}_q[X] \) a monic polynomial of degree \( d \geq 1 \). As efficient methods exist to reduce the factorization problem to the square-free case [20], in the following it is assumed without loss of generality that \( f \) is square-free, i.e.,

\[
\tilde{f} = g_1 \cdot g_2 \cdots g_t
\]

with \( g_i \in \mathbb{F}_q[X] \) pairwise different monic irreducible polynomials of degree \( d_i \). The purpose is to determine the \( g_i \) from the given \( f \). There are different “linearization techniques” which transform this problem into linear algebra over \( \mathbb{F}_q \) and gcd calculations in \( \mathbb{F}_q[X] \). By the chinese remainder theorem there are \( e_i \in \mathbb{F}_q[X], \ i \in \{1, 2, ..., t\} \), with \( \deg e_i < d \) such that \( e_i \equiv 1 \mod g_i \) and \( e_i \equiv 0 \mod g_j \) for \( j \neq i \). Hence \( \mathcal{A} := \mathbb{F}_q[X]/(f) \cong \bigoplus_{i=1}^{t} \mathbb{F}_q[X]/(g_i) \cong \bigoplus_{i=1}^{t} \mathbb{F}_{q^{e_i}}(e_i + (f)). \) Let \( F \) denote the Frobenius operator \( h \mapsto h^q \) for \( h \in \mathbb{F}_q[X] \), and \( F \) the operator on \( \mathcal{A} \) induced by \( F \). Then the algebra of \( F \)-fixed points is \( \mathcal{B} := \bigoplus_{i=1}^{t} \mathbb{F}_q(e_i + (f)). \)

The classical factorization method of Berlekamp proceeds as follows: Let \( Q_f \in \mathbb{F}_q^{d \times d} \) be the matrix of \( F \) with respect to the standard basis \( S := \{1 + (f), X + (f), X^2 + (f), \ldots, X^{d-1} + (f)\} \), and \( \text{Id} = \text{Id}_d \) the \( d \times d \) identity matrix. Computing the nullspace of \( Q_f - \text{Id} \) gives a basis of \( \mathcal{B} \). Choosing \( 0 \neq h + (f) = (\sum_{i=1}^{t} c_i e_i) + (f) \in \mathcal{B} \) and taking \( \gcd(f, h) \) will lead to a nontrivial factorization of \( f \) whenever at least one \( c_i \) (but not all) vanishes.

In contrast to this, the starting point of Niederreiter’s algorithm is the following modular linear differential equation in the field \( \mathbb{F}_q(X) \) of rational functions:

\[
H^{(q-1)}(y) - y^q = 0.
\]

Here \( H^{(k)} \) is the Hasse-Teichmüller derivative, defined in the field of formal Laurent series in \( X^{-1} \) over \( \mathbb{F}_q \):

\[
H^{(k)} \left( \sum_{n=j}^{\infty} s_n X^{-n} \right) = \sum_{n=j}^{\infty} \binom{-n}{k} s_n X^{-n-k}, \ j \in \mathbb{Z}.
\]
The following theorem gives the connection to polynomial factorization:

**Theorem 2.1** (Niederreiter). The set of solutions of equation (1) with fixed denominator $f$ is an $\mathbb{F}_q$-space $\mathcal{L}$ of elements of degree $\leq -1$:

$$\mathcal{L} = \left\{ \frac{g'_1, g'_2, \ldots, g'_t}{g_1, g_2, \ldots, g_t} \right\}.$$  

These solutions satisfy the equation $f^q \cdot H^{(q-1)}(\frac{h}{q}) = h^q$. Both sides of this equation are polynomials in $X^q$; hence a comparison of coefficients at $X^q$ yields a system of linear equations with coefficient matrix $N_f$ such that, for $h = h_0 + h_1 X + \ldots + h_{d-1} X^{d-1}$ and $h := (h_0, \ldots, h_{d-1})$,

$$h / f \in \mathcal{L} \iff (N_f - \text{Id}) \mathbf{h}^T = 0.$$  

From this it is possible to compute a basis of $\mathcal{N} := f \mathcal{L}$ mod $(f) \subseteq \mathcal{A}$ without knowing the irreducible factors $g_i$ of $f$.

Recently, D. Wan introduced a new operator which is closely related to the Niederreiter operator and also leads to an algorithm for polynomial factorization [13]. Before describing this algorithm, the details of the arguments that lead to an explicit formula for the entries of $N_f$ from [15] are given. The reason for this is that they also reveal a formula for the entries of the “Wan-matrix” (see also [16]): As in [6], it can be shown that $f^q H^{(q-1)}(h/f) = H^{(q-1)}(f^{q-1} h)$. Define $j := f^{q-1} h$.

Notice that $\text{deg}(j) \leq dq - 1$, i.e.,

$$j = \begin{array}{cccccc}
  j_0 & + & j_1 X & + & \ldots & + & j_{d-1} X^{d-1} \\
  + & j_q X^q & + & j_{q+1} X^{q+1} & + & \ldots & + & j_{2q-1} X^{2q-1} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
  + & j_{(d-1)q} X^{(d-1)q} & + & j_{(d-1)q+1} X^{(d-1)q+1} & + & \ldots & + & j_{dq-1} X^{dq-1} 
\end{array}.$$  

Now the definition of the Hasse-Teichmüller derivative and the relations

$$\left( \begin{array}{c} n + q - 1 \\ q - 1 \end{array} \right) \equiv \begin{cases} 1 \mod p & \text{for } n \equiv 0 \mod q, \\ 0 \mod p & \text{for } n \not\equiv 0 \mod q. \end{cases}$$

give

$$H^{(q-1)}(j) = j_{q-1} + j_{2q-1} X^q + \ldots + j_{dq-1} X^{(d-1)q},$$

from which the coefficients of $N_f$ can be read off: Let $f^{q-1} =: a = a_0 + a_1 X + \ldots + X^{d(q-1)}$ and define $a_j := 0$ for $j < 0$ or $j > d(q-1)$; then

$$N_f = \begin{pmatrix}
  a_{q-1} & a_{q-2} & \cdots & a_{q-d+1} & a_{q-d} \\
  a_{2q-1} & a_{2q-2} & \cdots & a_{2q-d+1} & a_{2q-d} \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  a_{(d-1)q-1} & a_{(d-1)q-2} & \cdots & a_{(d-1)q-d+1} & a_{(d-1)q-d} \\
  0 & 0 & \cdots & 0 & 1
\end{pmatrix}.$$  

Wan defined the linear map $\Delta : \mathbb{F}_q[X] \rightarrow \mathbb{F}_q[X]$ by

$$\Delta(X^k) := \begin{cases} X^k & \text{for } k \equiv 0 \mod q, \\ 0 & \text{for } k \not\equiv 0 \mod q. \end{cases}$$
From the discussion above, it follows directly that equation (1) can also be written as

\[ \Delta \left( H^{(q-1)}(f^{q-1}h) \right) = h. \]

Note that the operators \( h \mapsto \Delta \left( H^{(q-1)}(f^{q-1}h) \right) \) and \( h \mapsto \Delta \left( f^{q-1}h \right) \) stabilize the ideal \( (f) \subseteq \mathbb{F}_q[X] \); hence they induce operators \( N \) and \( W \) on \( A \) respectively. The matrix \( N_f \) describes \( N \) with respect to the basis \( S \); similarly \( W_f \) is defined to be the matrix of \( W \) with respect to this basis. Notice that \( \mathcal{N} = \ker(N - \text{Id}) \), while \( B = \ker(F - \text{Id}) \).

Niederreiter and Göttfert showed in Corollary 3.1 in [15] that the Niederreiter and the Berlekamp matrices are similar, i.e., a non-singular matrix \( G_f \in \mathbb{F}_q^{d \times d} \) exists such that \( N_f = G_fQ_fG_f^{-1} \). Their proof is based on the analysis of characteristic linear recurring sequences connected to \( N_f \). The following is a different proof of this fact:

**Proposition 2.2.** Define the operator \( N := \Delta(H^{(q-1)}(f^{q-1} \cdot \cdot \cdot)) \) on \( A \). Then \( N \) is conjugate in \( GL(A) \) to the Frobenius-automorphism \( F \).

**Proof.** In [4] it has been shown that for any field \( K \) and \( g, h \in K[X] \) with \( \deg h < \deg g \) one has

\[ \Delta (f^{q-1}h) = h. \]

Note that \( N = F^{-1} \circ H^{(q-1)}(\cdot/f) \circ (\cdot/f^q) \). Combining this with equation (5) yields

\[ N = F^{-1} \circ (f')^{q-1} = (f') \circ F^{-1} \circ (f')^{-1}, \]

which shows that \( N \) is conjugate to \( F^{-1} \). Clearly \( F \) and \( F^{-1} \) are conjugate to each other, as they are described by permutation matrices with respect to normal bases for the summands of \( A \cong \bigoplus_{i=1}^{d} \mathbb{F}_{q^i} \).

Wan showed that the Hasse-Teichmüller-derivative can be removed from equation (1), i.e., the solutions of

\[ \Delta(f^{q-1}h) = h \]

can still be used to factor the polynomial \( f \). To see how, notice that from [2], [3], and the definition of \( \Delta \) it follows that

\[ \Delta(f^{q-1}hX) = X \Delta(H^{(q-1)}(f^{q-1}h)). \]

From this the next theorem follows immediately.

**Theorem 2.3.** Assume that \( X \) does not divide \( f \), i.e., \( X = X + (f) \) is a unit in \( A \), and let \( W \) be the kernel of the linear operator \( W - \text{Id} = \Delta(f^{q-1}) - \text{Id} \) on \( A \). Let \( f^{q-1} := a = a_0 + a_1X + \cdots + X^{d(q-1)} \) and define \( a_j := 0 \) for \( j < 0 \) or \( j > d(q-1) \). Then

\[ \bar{h} \in W \iff (W_f - \text{Id})h^T = 0, \]

with

\[
W_f = \begin{pmatrix}
a_0 & 0 & \cdots & 0 & 0 \\
a_q & a_{q-1} & \cdots & a_{q-d+2} & a_{q-d+1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
a_{(d-2)q} & a_{(d-2)q-1} & \cdots & a_{(d-2)q-d+2} & a_{(d-2)q-d+1} \\
a_{(d-1)q} & a_{(d-1)q-1} & \cdots & a_{(d-1)q-d+2} & a_{(d-1)q-d+1}
\end{pmatrix}
\]
Moreover,  
\[ W = (\cdot X) \circ N \circ (\cdot X^{-1}) \]
and  
\[ W = N \cdot \tilde{X} = B \cdot \tilde{f}' \cdot \tilde{X}. \]
In particular, \( N \), \( W \), \( F \) and the corresponding matrices have the same minimal polynomial.

Once an element \( h \) of the Berlekamp, Niederreiter or Wan subspace is found, this element can be used to factor the polynomial \( f \). In the following, let \( u = 1 \in \mathbb{F}_q[X] \) if \( h \in \mathcal{B} \), \( u = f' \in \mathbb{F}_q[X] \) if \( h \in \mathcal{N} \) and \( u = f' \cdot X \in \mathbb{F}_q[X] \) if \( h \in \mathcal{W} \). When the field size is small, one can use the well-known fact that  
\[ f = \prod_{\alpha \in \mathbb{F}_q} \gcd(f, h - \alpha u), \]
involving a non-trivial factorization as long as \( h \) is not an element of \( u\mathbb{F}_q \). For large fields of odd characteristic an efficient probabilistic method due to Cantor and Zassenhaus [3] can be applied, where computing  
\[ \gcd(f, h^{\frac{2^i-1}{2}} - u^{\frac{2^i-1}{2}}) \]
leads to a non-trivial factorization with high probability.

### 3. The black-box method

In 1986 Wiedemann introduced an algorithm for solving sparse linear systems of equations over finite fields [19]. The algorithm can be applied to produce solutions \( x \in \mathbb{F}_{q^d}^d \) of  
\[ Bx^T = 0, \]
with \( B \in \mathbb{F}_{q^d}^{d \times d} \) and is of “Las Vegas” type, which means that either failure or the correct answer is returned upon termination. It requires \( O(d) \) multiplications of the matrix \( B \) with a vector in \( \mathbb{F}_{q^d}^d \) and \( O(d^2 \log d) \) additional field operations. It was pointed out in [19] that the method is not only efficient for sparse matrices; the only requirement is that the matrix-vector product can be computed efficiently. This leads to the so-called black-box model. The black-box accepts a vector \( w \in \mathbb{F}_{q^d}^d \) and returns \( Bw^T \) (see Figure 1). An important advantage over classical Gaussian elimination is that the matrix does not have to be stored densely in memory.

The following proposition describes what the black-box operation looks like if a polynomial over \( \mathbb{F}_q \) is factored with Niederreiter’s algorithm (see also [16]). In this case \( B = N_f - \text{Id} \), i.e., the black-box accepts the vector \( w = (w_0, w_1, \ldots, w_{d-1}) \) and returns the vector \( z = (z_0, z_1, \ldots, z_{d-1}) \) such that \( z^T = (N_f - \text{Id})w^T \).

![Figure 1. The black-box representation of B](image-url)
Corollary 3.2 (Black-box Wan operation). From this the following corollary follows immediately.

With the notations as in Proposition 3.1, the black-box Wan operation can be computed by

\begin{equation}
\tag{10}
z(X) = w_0(X)a_0(X) + \sum_{i=1}^{\min\{d-1,q-1\}} w_i(X)a_{q-1-i}(X) - w(X),
\end{equation}

where \(a(X) = f(X) = f_0(X^2) + Xf_1(X^2)\), implying that this operation can also be computed using \(O(d \log^{\log_2 3} d)\) field operations if Cantor multiplication is used.
Consider if and only if this procedure is applied recursively, eventually one will reach an element \( v \) with

\[
\alpha_i = u B^i v^T, \quad i = 0, 1, \ldots, 2d - 1.
\]

With the help of the Berlekamp/Massey algorithm \([11]\) a minimal characteristic polynomial of a linear feedback shift register (LFSR) which generates the sequence can be computed. This polynomial

\[
\mu_{u, B, v}(Y) = c_0 Y^d + c_{d-1} Y^{d-1} + \cdots + c_k Y^k
\]

with \( c_k \neq 0 \) divides the minimal polynomial \( \mu_{B,v} \) of the corresponding vector sequence \((B^i v^T)_i\), which itself divides the minimal polynomial \( \mu_B \) of the matrix \( B \). Lower bounds on the probability that \( \mu_{u, B, v} \) and \( \mu_{B, v} \) (respectively \( \mu_{B,v} \) and \( \mu_B \)) are equal can be found in \([19]\) and \([9]\), but software implementations indicate that there is still room for improving these bounds. Assume that \( \mu_{u, B, v} = \mu_B \); if \( k = 0 \) then \( B \) is non-singular, else define \( \tilde{\mu}_B(Y) := \mu_B(Y)/Y^k \). Compute \( \tilde{\mu}_B(B)v^T \) and the sequence

\[
B^i \tilde{\mu}_B(B)v^T, \quad i = 1, 2, \ldots, k - 1.
\]

The last vector \( x \) of this sequence is then a nontrivial solution of \((11)\), since

\[
B B^{k-1} \tilde{\mu}_B(B)v^T = 0.
\]

A solution of \( Bx^T = 0 \) is found whenever

\[
\mu_{B,v}(Y) = c_0 Y^d + c_{d-1} Y^{d-1} + \cdots + c_k Y^k
\]

with \( c_k \neq 0 \) and \( k \geq 1 \) (see \([11]\)). If \( k = 0 \), then \( \mu_{B,v}(Y) \) is a proper divisor of \( \mu_B(Y) \), and the procedure above can be repeated with some \( 0 \neq \mu_{B,v}(B)v^T := v' \). With this choice for \( v' \), it follows that \( \mu_{B,v'}(Y) \) divides \( \mu_B(Y)/\mu_{B,v}(Y) \). If this procedure is applied recursively, eventually one will reach an element \( v'' \) with \( \mu_{B,v''}(0) = 0 \). Note also that in case \( B \) is non-singular, one will eventually find \( \mu_B(Y) \) by applying this procedure. In the following it is shown that a random element \( v \in \mathbb{F}_q^d \) very likely will have a minimal polynomial divisible by \( Y \), even if the field size \( q \) is small:

**Lemma 3.3.** Let \( B \in \mathbb{F}_q^{d \times d} \) be singular and let \( \mathbb{F}_q = V_0 \oplus V_1 \) be the “Fitting decomposition”, i.e., \( B \) acts nilpotently on \( V_0 \) and invertibly on \( V_1 \).

Then \( Y \) does not divide \( \mu_{B,v}(Y) \) if and only if \( v \in V_1 \). In particular, the probability of \( \mu_{B,v}(0) = 0 \) for a randomly chosen element \( v \) is

\[
> 1 - \frac{1}{q^{\dim V_0}} \geq 1 - \frac{1}{\ker B}.
\]

**Proof.** Let \( \mu_B(Y) = \mu_{B,v}(Y) = Y^k \cdot \tilde{\mu}(Y) \) be such that \( Y \) does not divide \( \tilde{\mu}_B(Y) \). Consider \( \mathbb{F}_q^d \) as an \( \mathbb{F}_q[Z] \)-module with \( Z \) acting like \( B \) and the Fitting decomposition \( \mathbb{F}_q^d = V_0 \oplus V_1 \) with \( V_0 = \ker Z^k \) and \( V_1 = \tilde{\mu}_B(Z) \). So \( Z \) does not divide \( \mu_B(Z) \) if and only if \( v \) has a trivial component in \( V_0 \) and lies in \( V_1 \). Hence the probability for a randomly chosen element \( v \) to have a non-trivial component in \( V_0 \) is

\[
1 - \frac{q^{\dim V_1} - 1}{q^d - 1} > 1 - \frac{1}{q^{\dim V_0}} \geq 1 - \frac{1}{\ker B}.
\]

\[ \square \]
Notice that \( \dim V_0 \geq \dim \ker B \), so in polynomial factorization, where, e.g., 
\( B = N_f - \text{Id} \) or \( B = W_f - \text{Id} \), the chance of finding a “useful” element \( v \) increases 
with the number of irreducible factors of \( f (= \dim \ker N_f - \text{Id}) \).

When “explicit” linear algebra based on Gaussian elimination is used, a basis for 
the nullspace is found. The dimension of this basis coincides with the number of 
irreducible factors. However, the “implicit” method based on Wiedemann’s algo-

rithm does not provide this information, and therefore a condition for irreducibility 
is needed for the termination of the algorithm. Recall that for a square-free poly-
nomial \( f \), the Berlekamp, Niederreiter and Wan matrices all have the same minimal 
polynomial. If \( f = g_1 g_2 \ldots g_t \) with \( d_i = \deg(g_i) \), then this minimal polynomial is 
given by (see [3])

\[
\mu_B + \text{Id}(Y) = \mu_B(Y - 1) = \text{lcm}_d(Y^{d_i} - 1).
\]

The statements in the following lemma can be found in [10] for the Berlekamp 
matrix, and can be applied directly to the Niederreiter matrix and the Wan matrix.

**Lemma 3.4.** For a square-free polynomial \( f \) the following statements are equiva-

tent:

1. The polynomial \( f \) is irreducible.
2. The minimal polynomial \( \mu_B(Y) \) equals \( (Y + 1)^d - 1 \).
3. The minimal polynomial \( \mu_B(Y) \) is of degree \( d \).

For completeness the pseudo-code of this nullspace algorithm for polynomial 

factorization is given. Note that the matrix \( B \) is always singular in this case, as 
the dimension of its nullspace coincides with the number of irreducible factors of 
\( f \). However, recall from Section 2 that the elements in \( uF_q \) of this nullspace (with 
\( u = 1 \in F_q[X] \) if \( B = Q_f - \text{Id} \), \( u = f' \in F_q[X] \) if \( B = N_f - \text{Id} \) and \( u = f' \cdot X \in F_q[X] \) 
if \( B = W_f - \text{Id} \)) are not useful for factoring \( f \). The algorithm therefore returns either 
a “useful” element of the nullspace or the zero vector if the polynomial is found 

in this case). In practice it turns out that many iterations, i.e., many choices for 

\( u \) and \( v \), can be needed for this. The two statements in the following lemma, 

which depend on the special form of \( \mu_B(Y) \) as given in [12], are often useful in this case.

If after a number of choices for \( u \) and \( v \) neither a useful nullspace element nor 
the complete minimal polynomial \( \mu_B(Y) \) can be found, one can compute gcd’s of
nullspace algorithm

\begin{verbatim}
vector nullspace(integer d)
{
  \tau_B(Y) \leftarrow 1
  do
  \{ \tilde{\tau}_B(Y) \leftarrow \frac{\tau_B(Y)}{Y_{\mu}} \quad \text{(such that } \tilde{\tau}_B(0) \neq 0) \}
  do
  { choose a random \( v \) in \( \mathbb{F}_q^d \)
    \( v^T \leftarrow \tilde{\tau}_B(B)v^T \)
  } while (\( v = 0 \))
  \mu_{B, v}(Y) \leftarrow 1
  do
  { choose a random \( u \) in \( \mathbb{F}_q^d \)
    compute \( S = (uB^i v^T)_{i=0,...,2(d-\deg(\tau_B))} \)
    \mu_{u, B, v}(Y) \leftarrow \text{Berlekamp-Massey}(S; n - \deg(\tilde{\tau}_B))
    \mu_{B, v}(Y) \leftarrow \text{lcm}(\mu_{B, v}(Y), \mu_{u, B, v}(Y))
  } while (\( \mu_{B, v}(B) v^T \neq 0 \))
  if (\( Y \mid \mu_{B, v}(Y) \))
    \{ \( x^T \leftarrow \frac{\mu_{B, v}(Y)}{Y} (B)v^T \)
      if (\( x \) not in \( u\mathbb{F}_q \)) return \( x \)
    \}
  \tau_B(Y) \leftarrow \text{lcm}(\tau_B(Y), \mu_{B, v}(Y))
  do
  { \text{while (} \deg(\tau_B) < d \text{) }
    return 0
  }
}
\end{verbatim}

\( \tau(B) \) with a number of (precomputed) cyclotomic polynomials and try to find the missing factors of \( \mu_B(Y)/\tau_B(Y) \) before applying the irreducibility test (step 25). If \( \tau_B(Y) = \mu_B(Y) \) after these computations, then further iterations are avoided.

**Lemma 3.5.** Let \( \Phi_m(Y) \) denote the cyclotomic polynomial of order \( m \) over \( \mathbb{F}_q \).

1. If \( \gcd(\Phi_m(Y+1), \mu_B(Y)) \neq 1 \), then \( \Phi_l(Y+1) \) divides \( \mu_B(Y) \) for every \( l \mid m \).
2. The irreducible factors of \( \mu_B \) occur in \( p \)-powers.

**Proof.** Note that \( \gcd(\Phi_m(Y), \Phi_s(Y)) = 1 \) for \( m \neq s \), as their roots are of different order. If \( d_i = d_i'p^k \) with \( p \) not dividing \( d_i' \), then from \([12]\) it follows that

\[
\mu_B(Y) = \text{lcm}_{d_i}((Y+1)^{d_i} - 1) = \text{lcm}_{d_i}(((Y+1)^{d_i} - 1)^{p^k})
= \text{lcm}_{d_i'}\left( \prod_{s \mid d_i'} \Phi_{s}(Y + 1)^{p^k} \right),
\]

which proves the second statement. If \( D \) denotes the set of divisors of the \( d_i' \)'s, then from this it also follows that, up to \( p \)-powers \( \mu_B(Y) \) equals \( \prod_{s \in D} \Phi_{s}(Y + 1) \). If \( \gcd(\Phi_m(Y+1), \mu_B(Y)) \neq 1 \), then there is an irreducible \( h \) dividing both \( \Phi_m(Y+1) \)
and $\mu_B(Y)$; hence $h$ divides $\Phi_s(Y + 1)$ with $s \mid d'_i$ for some $i$. This implies that $\gcd(\Phi_m(Y + 1), \Phi_s(Y + 1)) \neq 1$, and consequently $m = s$. Every divisor $l$ of $m$ also divides $d'_i$, so $l \in D$ and therefore $\Phi_l(Y + 1)$ divides $\mu_B(Y)$.

Remark 3.6. If a square-free polynomial is to be factored completely, one can proceed as follows. Once a non-trivial factorization $f = g \cdot h$ with $\deg(g) \geq \deg(h)$ is found by using some divisor $\tau_{B_f}(Y)$ of the minimal polynomial $\mu_{B_f}(Y)$, find the complete factorization of the (smallest) factor $h$. The factorization pattern (i.e., degrees of the irreducible factors) of $h$ directly gives the minimal polynomial $\mu_{B_h}(Y)$ (see (12)). This information can be used to compute a divisor

$$\tau_{B_h}(Y) := \frac{\tau_{B_f}(Y)}{\gcd(\tau_{B_f}(Y), \mu_{B_h}(Y))}$$

of the minimal polynomial $\mu_{B_h}(Y)$, as $\mu_{B_f}(Y) = \text{lcm}(\mu_{B_a}(Y), \mu_{B_h}(Y))$. In practical situations, one obtains that way a large factor of the minimal polynomial of $B_g$ basically for free. Of course this procedure can be applied recursively.

4. Benchmarks

The black-box Niederreiter algorithm described in the previous section is well-suited for factoring high-degree polynomials over small finite fields. It was implemented for factoring polynomials over $\mathbb{F}_2$, which is of particular interest for practical applications. All programs were written in the C programming language and are based on the implementations for fast polynomial arithmetic as described in [17]. These implementations were further optimized for speed, especially the Cantor multiplication, which is the basic operation in the black-box Niederreiter algorithm. From the previous sections, it is clear that an implementation of the black-box Wan algorithm would lead to similar benchmarks, as it is closely related to the Niederreiter algorithm.

All running times contained in the tables are in days-hours-minutes. The polynomials to be factored were generated in a pseudo-random way and are therefore “dense” polynomials. Table 1 contains the running times for the complete factorization of five different polynomials of degree 64000. These times were obtained using a Pentium III processor rated at 450 MHz. From this table it can be seen that the factorization time depends on the factorization pattern (mainly on the number of factors) of the polynomial to be factored, with an average time of three hours and 37 minutes. This is in contrast to implementations based on “explicit” linear algebra, as described, e.g., in [17].

Table 2 depicts the running times for the complete factorization of polynomials of growing degree. For each polynomial, this table also contains the time needed for the first execution of steps 15 and 18 of the algorithm (i.e., the generation of one sequence with a length of twice the degree of the polynomial and the evaluation of $\mu_{B,Y}(B)\nu^T$). It can be seen that this can take more than half of the time needed for the complete factorization. All benchmarks were obtained using a Pentium III processor rated at 450 Mhz, with the exception of the degree 1024000 polynomial, which was factored using a Pentium III processor rated at 500 Mhz.

In Table 3 the performance of the sequential implementation of the factorization algorithm is compared with a parallel implementation. In this parallel implementation, the computation of the two polynomial multiplications in the Cantor multiplication (see (9)) are distributed over two different processors. Moreover, a
multiplication algorithm derived from Cantor’s algorithm which allows parallelization was used (as described in [7]). The times for the parallel implementation were obtained by using a Siemens hpc-line Cluster with Pentium III processors, each rated at 400 Mhz. For the computation of the efficiency, an experimentally measured factor of 1.15 was taken into account to compensate for the slightly lower processor speed of the cluster. Although the performance of this algorithm does
Table 3. Comparison of sequential vs. parallel implementation

<table>
<thead>
<tr>
<th>Degree of Polynomial and factorization pattern</th>
<th>Sequential Total time</th>
<th>#procs</th>
<th>Parallel Total time</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>128000=83622+26801+13682+2761+739+262+37+21+14+13</td>
<td>00d 15h 00min</td>
<td>4 + 4 = 8</td>
<td>00d 03h 12min</td>
<td>67%</td>
</tr>
<tr>
<td>(15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256000=119302+73994+56536+2445+2420+449+273+214+187+129+45+6</td>
<td>02d 16h 49min</td>
<td>8 + 8 = 16</td>
<td>00d 09h 00min</td>
<td>52%</td>
</tr>
<tr>
<td>(12)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>512000=265932+242027+3751+206+42+31+10+1</td>
<td>09d 06h 45min</td>
<td>8 + 8 = 16</td>
<td>01d 06h 43min</td>
<td>52%</td>
</tr>
<tr>
<td>(8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

not scale up as well with the number of processors used as the parallel implementation of the “explicit” approach (see [17]), it does decrease the running times with a good efficiency when a small number of processors are used.

The implementations show that the black-box Niederreiter algorithm can be used for factoring high-degree polynomials over \( F_2 \) in a reasonable amount of time on a (small cluster of) personal computer(s). So when a large cluster is not available or the amount of memory is limited, the black-box approach is certainly a good alternative for the implementation of the Niederreiter algorithm based on “explicit” linear algebra as described in [17].

ACKNOWLEDGMENT

The authors are grateful to the referee for his/her valuable remarks.

REFERENCES


Institute of Mathematics and Statistics, University of Kent, Canterbury, CT2 7NF, England

E-mail address: p.fleischmann@ukc.ac.uk

Westdeutsche Landesbank, Düsseldorf/Münster, Germany

E-mail address: markus.holder@epost.de

Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Current address: Philips Semiconductors, B.V., P.O. Box 218, 5600 MD Eindhoven, The Netherlands

E-mail address: peter.roelse@philips.com

License or copyright restrictions may apply to redistribution; see http://www.ams.org/journal-terms-of-use