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 in 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 [14].

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.

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

1991 *Mathematics Subject Classification.* Primary 11-04, 11T06, 11Y16.
Key words and phrases. Finite fields, polynomial factorization, implicit linear algebra.

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/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 an algorithm proposed recently by Wan [18], which is closely related to the Niederreiter algorithm. In Section 3 an overview of the black-box approach is given, including the algorithm used to find elements of the nullspace in Niederreiter's (or Wan's) algorithm. Section 4 contains the benchmarks for an implementation of this algorithm in the C programming language. Both run-times for a sequential implementation as well as 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.

$$f = g_1 \cdot g_2 \dots g_t$$

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 $\mathcal{A} := \mathbb{F}_q[X]/(f) \cong \oplus_{i=1}^{t} \mathbb{F}_q[X]/(g_i) \cong \oplus_{i=1}^{t} \mathbb{F}_{q^{d_i}}(e_i + (f))$. 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 $\mathcal{B} := \oplus_{i=1}^{t} \mathbb{F}_q(e_i + (f))$.

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 $S := \{1 + (f), X + (f), X^2 + (f), \dots, X^{d-1} + (f)\}$ and $\mathbf{Id} = \mathbf{Id}_d$ the $d \times d$ identity matrix. Computing the nullspace of $Q_f - \mathbf{Id}$ 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 of rational functions $\mathbb{F}_q(X)$:

$$H^{(q-1)}(y) - y^q = 0. \tag{1}$$

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

$$H^{(k)}(\sum_{n=j}^{\infty} s_n X^{-n}) = \sum_{n=j}^{\infty} \binom{-n}{k} s_n X^{-n-k}, \ j \in \mathbf{Z}.$$

The following theorem gives the connection to polynomial factorization:

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

$$\mathcal{L} = < \frac{g_1{}'}{g_1}, \frac{g_2{}'}{g_2}, \dots, \frac{g_t{}'}{g_t} >_{\mathbb{F}_q}$$

These solutions satisfy the equation $f^q \cdot H^{(q-1)}(\frac{h}{f}) = h^q$. 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 $h = h_0 + h_1 X + \dots + h_{d-1} X^{d-1}$ and $\mathbf{h} := (h_0, \dots, h_{d-1})$:

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

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 as well leads to an algorithm for polynomial factorization [18]. Before describing this algorithm, the details of the arguments that lead to an explicit formula for the entries of N_f from [5] 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 [5] 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 $\deg(j) \leq dq - 1$, i.e.

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

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

$$\binom{n+q-1}{q-1} \equiv \begin{cases} 1 \mod p & \text{for } n \equiv 0 \mod q, \\ 0 \mod p & \text{for } n \not\equiv 0 \mod q. \end{cases}$$

give

$$H^{(q-1)}(j) = j_{q-1} + j_{2q-1}X^q + \dots + j_{dq-1}X^{(d-1)q}, \tag{3}$$

from which the coefficients of N_f can be read off: Let $f^{q-1} =: a = a_0 + a_1 X + \dots + X^{d(q-1)}$ 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 $\Delta : \mathbb{F}_q[X] \to \mathbb{F}_q[X]$ by

$$\Delta(X^k) := \begin{cases} X^{\frac{k}{q}} & \text{for } k \equiv 0 \mod q, \\ 0 & \text{for } k \not\equiv 0 \mod q. \end{cases}$$

From the discussion above, it follows directly that Equation 1 can also be written as

$$\Delta\left(H^{(q-1)}(f^{q-1}h)\right) = h. \tag{4}$$

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 $\mathcal{A}$ 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 $\mathcal{N} = \ker(\mathsf{N} - \mathbf{Id})$ while $\mathcal{B} = \ker(\mathsf{F} - \mathbf{Id})$.

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 $N_f = G_f Q_f G_f^{-1}$. 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* $\mathsf{N} := \Delta(H^{(q-1)}(f^{q-1}\cdot))$ *on* $\mathcal{A}$. *Then* N *is conjugate in* $GL(\mathcal{A})$ *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

$$H^{(k)}\left(\frac{h}{g}\right) \cdot g^{k+1} \;\equiv\; (-1)^k \cdot h \cdot (g')^k \mod g. \tag{5}$$

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

$$\mathsf{N} = \mathsf{F}^{-1} \circ (\cdot (f')^{q-1}) = (\cdot f') \circ \mathsf{F}^{-1} \circ (\cdot (f')^{-1}),$$

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 $\mathcal{A} \cong \oplus_{i=1}^{t} \mathbb{F}_{q^{d_i}} \bar{e}_i$. □

Wan showed that the Hasse-Teichmüller-derivative can be removed from Equation 4, 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 Equation 2, Equation 3 and the definition of Δ 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.* $\bar{X} = X + (f)$ *is a unit in* $\mathcal{A}$ *and let* $\mathcal{W}$ *be the kernel of the linear operator* $\mathsf{W} - \mathbf{Id} = \Delta(f^{q-1}\cdot) - \mathbf{Id}$ *on* $\mathcal{A}$. *Let* $f^{q-1} =: a = a_0 + a_1 X + \cdots + X^{d(q-1)}$ *and define* $a_j := 0$ *for* $j < 0$ *or* $j > d(q-1)$, *then*

$$\bar{h} \in \mathcal{W} \iff (W_f - \mathbf{Id})\mathbf{h}^T = \mathbf{0},$$

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

$$\mathsf{W} = (\cdot\bar{X}) \circ \mathsf{N} \circ (\cdot\bar{X}^{-1}).$$

and

$$\mathcal{W} = \mathcal{N} \cdot \bar{X} = \mathcal{B} \cdot \bar{f}' \cdot \bar{X}.$$

In particular N, W, F *and the corresponding matrices have the same minimal polynomial.*

Once an element h of either 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

$$\gcd(f, h^{\frac{q-1}{2}} - u^{\frac{q-1}{2}})$$

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 $\mathbf{x} \in \mathbb{F}_q^d$ of

$$B\mathbf{x}^T = 0, \tag{6}$$

with $B \in \mathbb{F}_q^{d \times d}$ 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 $\mathbb{F}_q^d$ and $O(d^2 \log d)$ additional field operations. It was pointed out in [9] 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 $\mathbf{w} \in \mathbb{F}_q^d$ and returns $B\mathbf{w}^T$ (see Figure 1). An important advantage over classical Gaussian elimination is that the matrix does not have to be stored densely in memory.

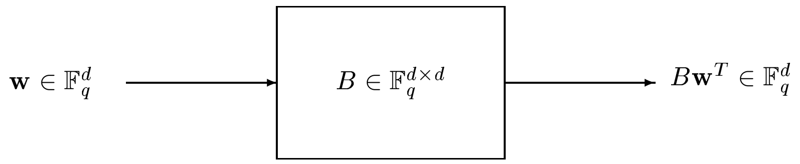

Figure 1. The black-box representation of B

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 $B = N_f - \mathbf{Id}$, i.e. the black-box accepts the vector $\mathbf{w} = (w_0 w_1 \ldots w_{d-1})$ and returns the vector $\mathbf{z} = (z_0 z_1 \ldots z_{d-1})$ such that $\mathbf{z}^T = (N_f - \mathbf{Id})\mathbf{w}^T$.

Proposition 3.1 (Black-box Niederreiter operation)**.** *Let f and a be defined as in Theorem 2.3. Define $w(X) = \sum_{i=0}^{d-1} w_i X^i$ and write $a(X) = \sum_{i=0}^{q-1} X^i \mathsf{a}_i(X^q)$ and $w(X) = \sum_{i=0}^{q-1} X^i \mathsf{w}_i(X^q)$, with $\mathsf{a}_i, \mathsf{w}_i \in \mathbb{F}_q[X]$. Then*

$$z(X) = \sum_{i=0}^{\min\{d-1,q-1\}} \mathsf{w}_i(X)\mathsf{a}_{q-1-i}(X) - w(X), \tag{7}$$

with $z(X) = \sum_{i=0}^{d-1} z_i X^i$.

Proof. Define $j(X) := a(X)w(X)$ and write $j(X) = \sum_{i=0}^{q-1} X^i \mathsf{j}_i(X^q)$ with $\mathsf{j}_i(X^q) \in \mathbb{F}_q[X]$. From Equation 3 and the definition of Δ it follows that

$$z(X) = \Delta\left(H^{(q-1)}(a(X)w(X))\right) - w(X) = \mathsf{j}_{q-1}(X) - w(X). \tag{8}$$

Note that the only terms that contribute to $\mathsf{j}_{q-1}(X^q)$ in $a(X)w(X)$ are given by $\mathsf{w}_i(X^q)\mathsf{a}_{q-1-i}(X^q)$ for $0 \le i \le q-1$. Finally observe that if $d-1 < q-1$ then $\mathsf{w}_i(X^q) = w_i = 0$ for $d-1 < i \le q-1$. $\square$

For the factorization of the polynomial $f \in \mathbb{F}_2[X]$ Equation 7 simplifies to

$$z(X) = \mathsf{w}_0(X)\mathsf{f}_1(X) + \mathsf{w}_1(X)\mathsf{f}_0(X) + w(X), \tag{9}$$

where $a(X) = f(X) = \mathsf{f}_0(X^2) + X\mathsf{f}_1(X^2)$ (see also [6]). This means that the black box call will roughly take the same time as two polynomial multiplications of half the degree of f. If Cantor multiplication [2] is used, this will even be faster since the evaluation of $\mathsf{f}_0(X)$ and $\mathsf{f}_1(X)$ can be precomputed. The complexity for Cantor multiplication in $\mathbb{F}_2[X]$ equals $O(d \log^{\log_2 3} d)$ field operations.

If a polynomial over $\mathbb{F}_q$ is factored with Wan's algorithm, then $B = W_f - \mathbf{Id}$, i.e. the black-box accepts the vector $\mathbf{w} = (w_0 w_1 \ldots w_{d-1})$ and returns the vector $\mathbf{z} = (z_0 z_1 \ldots z_{d-1})$ such that $\mathbf{z}^T = (W_f - \mathbf{Id})\mathbf{w}^T$. Note that Equation 8 is replaced by

$$z(X) = \Delta\left(a(X)w(X)\right) - w(X) = \mathsf{j}_0(X) - w(X).$$

From this the following corollary follows immediately.

Corollary 3.2 (Black-box Wan operation)**.** *With the notations as in Proposition 3.1, the black-box Wan operation can be computed by*

$$z(X) = \mathsf{w}_0(X)\mathsf{a}_0(X) + \sum_{i=1}^{\min\{d-1,q-1\}} \mathsf{w}_i(X)\mathsf{a}_{q-i}(X) - w(X). \tag{10}$$

From Equations 7 and 10, it is clear that both the Niederreiter and Wan black-box operation are comparable with respect to the number of operations needed. In practical implementations both operations will roughly take the same time. Notice also that for $f \in \mathbb{F}_2[X]$, Equation 10 simplifies to

$$z(X) = \mathsf{w}_0(X)\mathsf{f}_0(X) + \mathsf{w}_1(X)\mathsf{f}_1(X) + w(X),$$

where $a(X) = f(X) = \mathsf{f}_0(X^2) + X\mathsf{f}_1(X^2)$, implying that this operation can also be computed using $O(d \log^{\log_2 3} d)$ field operations if Cantor multiplication is used.

In the remainder of this section a brief description of the Wiedemann algorithm is given. Wiedemann's method first picks two random vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^d$ and computes the sequence of field elements

$$\alpha_i = \mathbf{u}B^i\mathbf{v}^T, \qquad i = 0, 1, \ldots, 2d-1.$$

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

$$\mu_{\mathbf{u},B,\mathbf{v}}(Y) = c_l Y^l + c_{l-1}Y^{l-1} + \cdots + c_k Y^k$$

with $c_k \neq 0$ divides the minimal polynomial $\mu_{B,\mathbf{v}}$ of the corresponding vector sequence $(B^i\mathbf{v}^T)_i$, which itself divides the minimal polynomial μ_B of the matrix B. Lower bounds on the probability that $\mu_{\mathbf{u},B,\mathbf{v}}$ and $\mu_{B,\mathbf{v}}$ respectively $\mu_{B,\mathbf{v}}$ and μ_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 $\mu_{\mathbf{u},B,\mathbf{v}} = \mu_B$; if $k = 0$ then B is non-singular, else define $\tilde{\mu}_B(Y) := \mu_B(Y)/Y^k$. Compute $\tilde{\mu}_B(B)\mathbf{v}^T$ and the sequence

$$B^i\tilde{\mu}_B(B)\mathbf{v}^T, \qquad i = 1, 2, \ldots, k-1.$$

The last vector $\mathbf{x}$ of this sequence is then a nontrivial solution of Equation 6, since

$$B\,\underbrace{B^{k-1}\tilde{\mu}_B(B)\mathbf{v}^T}_{\mathbf{x}^T\neq 0} = 0. \tag{11}$$

A solution of $B\mathbf{x}^T = 0$ is found whenever

$$\mu_{B,\mathbf{v}}(Y) = c_l Y^l + c_{l-1}Y^{l-1} + \cdots + c_k Y^k$$

with $c_k \neq 0$ and $k \geq 1$ (see Equation 11). If $k = 0$ then $\mu_{B,\mathbf{v}}(Y)$ is a proper divisor of $\mu_B(Y)$ and the procedure above can be repeated with some $0 \neq \mu_{B,\mathbf{v}}(B)\tilde{\mathbf{v}}^T =: \mathbf{v}'^T$. By making this choice for $\mathbf{v}'$, it follows that $\mu_{B,\mathbf{v}'}(Y)$ divides $\mu_B(Y)/\mu_{B,\mathbf{v}}(Y)$. If this procedure is applied recursively, eventually one will reach an element $\mathbf{v}''$ with $\mu_{B,\mathbf{v}''}(0) = 0$. 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 $\mathbf{v} \in \mathbb{F}_q^d$ 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 $\mu_{B,\mathbf{v}}(Y)$ if and only if $\mathbf{v} \in V_1$. In particular the probability of $\mu_{B,\mathbf{v}}(0) = 0$ for a randomly chosen element $\mathbf{v}$ is $> 1 - \frac{1}{q^{\dim V_0}} \geq 1 - \frac{1}{\ker B}$.

Proof. Let $\mu_B(Y) = \mu_{B,\mathbf{v}}(Y) = Y^k \cdot \tilde{\mu}_B(Y)$ such that Y does not divide $\tilde{\mu}_B(Y)$. Consider $\mathbb{F}_q^d$ as a $\mathbb{F}_q[Z]$-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 $V_1 = \ker \tilde{\mu}_B(Z)$. So Z does not divide $\mu_B(Z)$ if and only if $\mathbf{v}$ has a trivial component in V_0 and lies in V_1. Hence the probability for a randomly chosen element $\mathbf{v}$ to have a non-trivial component in V_0 is $1 - \frac{q^{\dim V_1}-1}{q^d-1} > 1 - \frac{1}{q^{\dim V_0}} \geq 1 - \frac{1}{\ker B}$. □

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

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 algorithm 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 polynomial f, the Berlekamp, Niederreiter and Wan matrix all have the same minimal polynomial. If $f = g_1 g_2 \dots g_t$ with $d_i = \deg(g_i)$ then this minimal polynomial is given by (see [8])

$$\mu_{B+\mathbf{Id}}(Y) = \mu_B(Y-1) = \mathrm{lcm}_{d_i}(Y^{d_i} - 1). \tag{12}$$

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 equivalent:*

(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 $u\mathbb{F}_q$ of this nullspace (with $u = 1 \in \mathbb{F}_q[x]$ if $B = Q_f - \mathbf{Id}$, $u = f' \in \mathbb{F}_q[x]$ if $B = N_f - \mathbf{Id}$ and $u = f' \cdot x \in \mathbb{F}_q[x]$ if $B = W_f - \mathbf{Id}$) 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 to be irreducible (i.e. the dimension of the nullspace equals one). The polynomial $\tau_B(Y)$ in this algorithm is a divisor of $\mu_B(y)$, the minimal polynomial of B. In Steps 5-10 a vector $\mathbf{v}$ is selected in such a way that $\mu_{B,\mathbf{v}}(Y) \mid \mu_B(Y)/\tilde{\tau}_B(Y)$. By dividing out this 'useless' part of $\mu_B(Y)$, it is assured that $\mu_{B,\mathbf{v}}(Y)$ has degree $\leq d - \deg(\tilde{\tau}_B)$. In Steps 11-18 the minimal polynomial $\mu_{B,\mathbf{v}}(Y)$ is computed for this choice of $\mathbf{v}$. If this polynomial is divisible by Y, a nullspace element is found (Step 21). Note that this element is already computed as it is an intermediate result of the computation needed in Step 18. However, this element might not be useful for factorization as discussed above (Step 22). If this is the case or if $\mu_{B,\mathbf{v}}(Y)$ is not divisible by Y, the polynomial $\tau_B(Y)$ is updated (Step 24) and the irreducibility test from Lemma 3.4 is applied (Step 25). If $\deg(\tau_B(Y)) = d$, then $\tau_B(Y) = \mu_B(Y)$ and the polynomial f is irreducible. If this is not the case, the procedure is repeated. In implementations, it is good practice to limit the number of attempts performed by the algorithm, and returning 'failure' if all of them were without success.

Note that for irreducible polynomials the algorithm terminates successfully as soon as the complete minimal polynomial $\mu_B(Y)$ is found (Lemma 3.3 is not useful in this case). In practice it turns out that many iterations, i.e. many choices for $\mathbf{u}$ and $\mathbf{v}$, 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 Equation 12, are often useful in this case. If after a number of choices for $\mathbf{u}$ and $\mathbf{v}$ neither a useful nullspace element nor the complete minimal polynomial $\mu_B(Y)$ can be found, one can compute gcd's of $\tau(B)$ with a number of (pre-computed) 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.

Nullspace algorithm

```
vector nullspace(integer d)
{
    τ_B(Y) ← 1
    do
    {
        τ̃_B(Y) ← τ_B(Y)/Y^m      (such that τ̃_B(0) ≠ 0)
        do
        {
            choose a random v in F_q^d
            v^T ← τ̃_B(B) v^T
        } while ( v = 0 )
        μ_{B,v}(Y) ← 1
        do
        {
            choose a random u in F_q^d
            compute S = (u B^i v^T)_{i=0,...,2(d−deg(τ̃_B))−1}
            μ_{u,B,v}(Y) ← Berlekamp-Massey(S, n − deg(τ̃_B))
            μ_{B,v}(Y) ← lcm(μ_{B,v}(Y), μ_{u,B,v}(Y))
        } while (μ_{B,v}(B) v^T ≠ 0)
        if ( Y | μ_{B,v}(Y) )
        {
            x^T ← (μ_{B,v}(Y)/Y)(B) v^T
            if (x not in uF_q) return x
        }
        τ_B(Y) ← lcm(τ_B(Y), μ_{B,v}(Y))
    } while ( deg(τ_B) < d )
    return 0
}
```

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 $l|m$.*
(2) *The irreducible factors of μ_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 $d_i = d_i' p^{k_i}$ with p not dividing d_i', then from Equation 12 it follows that

$$\mu_B(Y) = \mathrm{lcm}_{d_i}((Y+1)^{d_i}-1) = \mathrm{lcm}_{d_i'}(((Y+1)^{d_i'}-1)^{p^{k_i}}) = \mathrm{lcm}_{d_i'}(\prod_{s \,|\, d_i'} \Phi_s(Y+1)^{p^{k_i}}),$$

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 $\mu_B(Y)$. $\square$

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 Equation 12). This information can be used to compute a divisor $\tau_{B_g}(Y) := \tau_{B_f}(Y)/gcd(\tau_{B_f}(Y), \mu_{B_h}(Y))$ of the minimal polynomial $\mu_{B_g}(Y)$, as $\mu_{B_f}(Y) = \mathrm{lcm}(\mu_{B_g}(Y), \mu_{B_h}(Y))$. 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 e.g. described in [17].

Table 1. Benchmarks for degree 64000 polynomials

Factorization pattern	#factors	Total time
13742+13171+12498+ 10897+6207+3081+2003+ 1831+284+86+83+71+38+8	14	00d 04h 18min
37600+22985+3177+122+ 76+39+1	7	00d 03h 04min
51999+7870+3237+650+ 141+45+29+23+5+1	10	00d 04h 06min
63427+313+92+66+29+ 24+3+1	8	00d 02h 37min
57057+4190+2158+290+ 217+45+17+16+7+2+1	11	00d 04h 01min

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 Step 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 $\mu_{B,\mathbf{v}}(B)\mathbf{v}^T$). 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.

TABLE 2. Benchmarks sequential implementation

Degree of polynomial and factorization pattern	Benchmarks Sequence	Total time
128000 = 83622+26801+13682+ 2761+739+262+37+ 21+14+13+8+3+2+1 (15)	00d 08h 35min	00d 15h 00min
256000= 119302+73994+56536+ 2445+2420+449+273+ 214+187+129+45+6 (12)	01d 14h 49min	02d 16h 49min
512000= 265932+242027+ 3751+206+42+31+10+1 (8)	05d 13h 23min	09d 06h 45min
1024000= 730135+179324+ 62397+36547+7962+ 5015+2214+226+ 80+46+34+20 (12)	21d 02h 11min (PIII-500MHz)	60d 15h 27min (PIII–500MHz)

TABLE 3. Comparison sequential vs. parallel implementation

Degree of Polynomial and factorization pattern	Sequential Total time	Parallel #procs	Total time	Efficiency
128000= 83622+26801+13682+ 2761+739+262+37+ 21+14+13+8+3+2+1 (15)	00d 15h 00min	4 + 4 = 8	00d 03h 12min	67%
256000= 119302+73994+56536+ 2445+2420+449+273+ 214+187+129+45+6 (12)	02d 16h 49min	8 + 8 = 16	00d 09h 00min	52%
512000= 265932+242027+ 3751+206+42+ 31+10+1 (8)	09d 06h 45min	8 + 8 = 16	01d 06h 43min	52%

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 Equation 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 not scale up as well with the number of processors used as the parallel implementation of the 'explicit' approach (see [17]), it does allow to 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 $\mathbb{F}_2$ 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].

Acknowledgement

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

References

1. E.R. Berlekamp, *Factoring polynomials over finite fields*, Bell System Tech. J. **46** (1967), pp. 1853–1859.
2. D.G. Cantor, *On arithmetical algorithms over finite fields*, J. Combin. Theory Ser. A **50**, (1989), pp. 285–300.
3. D.G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Math. Comp. **36** (1981), pp. 587–592.
4. P. Fleischmann, *Connections between the algorithms of Berlekamp and Niederreiter for factoring polynomials over finite fields*, Linear Algebra Appl. **192** (1993), pp. 101–108.
5. P. Fleischmann and P. Roelse, *Comparative implementations of Berlekamp's and Niederreiter's polynomial factorization algorithms*, Finite Fields and their Applications (S. Cohen and H. Niederreiter, eds.), Cambridge University Press, 1996, pp. 73–84.
6. S. Gao and J. von zur Gathen, *Berlekamp's and Niederreiter's polynomial factorization algorithms*, Contemp. Math. **168** (1994), pp. 101–116.
7. M. Chr. Holder, *Arithmetik grossgradiger Polynome über endlichen Körpern - Theorie und Implementation*, PhD Thesis, Vorlesungen aus dem Fachbereich Mathematik der Universität GH Essen, Heft **28**, Universität GH Essen, 2000.
8. E. Kaltofen and A. Lobo, *Factoring high–degree polynomials by the black box Berlekamp algorithm*, ISSAC '94 (J. von zur Gathen and M. Giesbrecht, eds.), 1994, pp. 90–98.
9. E. Kaltofen and B.D. Saunders, *On Wiedemann's method of solving sparse linear systems*, Proc. AAECC-9, Springer Verlag, 1991, pp. 29-38.
10. E. Kaltofen and V. Shoup, *Subquadratic-time factoring of polynomials over finite fields*, Proc. 27th Annual ACM Symp. Theory of Comp., ACM Press, 1995, pp. 398–406.
11. J.L. Massey, *Shift–register synthesis and BCH decoding*, IEEE Trans. Inform. Theory **15** (1969), pp. 122-127.
12. M. Mignotte and D. Stefanescu, *Polynomials: An Algorithmic Approach*, Springer-Verlag, Singapore, 1999.
13. H. Niederreiter, *A new efficient factorization algorithm for polynomials over small finite fields*, Applicable Algebra Engrg. Comm. Comput. **4** (1993), pp. 81–87.
14. H. Niederreiter, *New deterministic factorization algorithms for polynomials over finite fields*, Contemp. Math. **168** (1994), pp. 251–268.
15. H. Niederreiter and R. Göttfert, *Factorizations of polynomials over finite fields and characteristic sequences*, J. Symbolic Comput. **16** (1993), pp. 401–412.
16. P. Roelse, *Linear Methods for Polynomial Factorization over Finite Fields - Theory and Implementations*, PhD Thesis, Vorlesungen aus dem Fachbereich Mathematik der Universität GH Essen, Heft **25**, Universität GH Essen, 1997.
17. P. Roelse, *Factoring high-degree polynomials over $\mathbb{F}_2$ with Niederreiter's algorithm on the IBM SP2*, Math. Comp. **68** (1999), no. 226, pp. 869–880.
18. D. Wan, *Computing zeta functions over finite fields*, Finite Fields: Theory, Applications, and Algorithms (R.C. Mullin and G.L. Mullen, eds.), Contemp. Math. **225**, American Mathematical Society, 1999, pp. 131–142.

19. D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986), pp. 54–62.
20. D.Y.Y. Yun, *On square-free decomposition algorithms*, Proc. ACM Symp. Symbolic and Algebraic Comput. (R.D. Jenks, ed.), 1976, pp. 26–35.

Institute of Mathematics and Statistics, University of Kent, Canterbury, CT2 7NF, UK
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
E-mail address: `p.l.a.roelse@tue.nl`