Randomized root finding over finite fields using tangent Graeffe transforms

Bruno Grenet, Joris van der Hoeven, Grégoire Lecerf

Randomized root finding over finite fields using tangent Graeffe transforms

Bruno Grenet
Laboratoire d'informatique, de robotique et de microélectronique de Montpellier
LIRMM, UMR 5506 CNRS, CC477
Université Montpellier 2
161, rue Ada
34095 Montpellier Cedex 5, France
bruno.grenet@lirmm.fr

Joris van der Hoeven, Grégoire Lecerf
Laboratoire d'informatique de l'École polytechnique
LIX, UMR 7161 CNRS
Campus de l'École polytechnique
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France
{vdhoeven,lecerf}@lix.polytechnique.fr

Version of January 16, 2015

ABSTRACT

Consider a finite field $\mathbb{F}_q$ whose multiplicative group has smooth cardinality. We study the problem of computing all roots of a polynomial that splits over $\mathbb{F}_q$, which was one of the bottlenecks for fast sparse interpolation in practice. We revisit and slightly improve existing algorithms and then present new randomized ones based on the Graeffe transform. We report on our implementation in the MATHEMAGIX computer algebra system, confirming that our ideas gain by an order of magnitude in practice.

Categories and Subject Descriptors

F.2.1 [**ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY**]: Numerical Algorithms and Problems–*Computations in finite fields*; G.4 [**MATHEMATICAL SOFTWARE**]: Algorithm design and analysis

General Terms

Algorithms, Theory

Keywords

Finite fields, polynomial root finding, algorithm, Mathemagix

1. INTRODUCTION

Let $\mathbb{F}_q$ represent the finite field with $q = p^k$ elements, where p is a prime number, and $k \geqslant 1$. Throughout this article, such a field is supposed to be described as a quotient of $\mathbb{F}_p[x]$ by a *monic* irreducible polynomial. Let $f \in \mathbb{F}_q[x]$ represent a *separable monic polynomial* of degree $d \geqslant 1$ which *splits* over $\mathbb{F}_q$, which means that all its irreducible factors have degree one and multiplicity one. In this article we are interested in computing all the roots of f.

1.1 Motivation

One of our interests in root finding came from the recent design of efficient algorithms to interpolate, into the standard monomial basis, polynomials that are given through evaluation functions. This task is briefly called *sparse interpolation*, and root finding often turns out to be a bottleneck, as reported in [19]. In fact, in this case, the ground field can be chosen to be $\mathbb{F}_p$ with $p = M\,2^m + 1$, and where 2^m is taken to be much larger than the number of terms to be discovered. In practice, to minimize the size of p, so that it fits a machine register, we take $M = O(\log p)$ as small as possible. A typical example is $p = 7 \cdot 2^{26} + 1$. We informally refer to such primes as *FFT primes*.

Root finding over prime finite fields critically occurs during the computation of integer and rational roots of polynomials in $\mathbb{Q}[x]$, both for dense and lacunary representations. Yet other applications concern cryptography and error correcting codes. Nevertheless, practical root finding has received only moderate attention so far, existing algorithms with good average complexity bounds often being sufficient [20, 21].

In this article, we focus on fast probabilistic root finding algorithms, targeting primarily FFT prime fields and, more generally, finite fields whose multiplicative group has smooth cardinality. At a second stage, we report on practical efficiency of our new algorithms within the MATHEMAGIX computer algebra system [18].

1.2 Notations and prerequisites

The *multiplicative group* $\mathbb{F}_q \setminus \{0\}$ of $\mathbb{F}_q$ is written $\mathbb{F}_q^*$. In order to simplify the presentation of complexity bounds, we use the *soft-Oh* notation: $f(n) \in \tilde{O}(g(n))$ means that $f(n) = g(n) \log^{O(1)} g(n)$ (we refer the reader to [11, Chapter 25, Section 7] for technical details). The least integer larger or equal to x is written $\lceil x \rceil$. The largest integer smaller or equal to x is written $\lfloor x \rfloor$. The *remainder* of g in the division by f is denoted by $g \operatorname{rem} f$.

We write $\mathsf{M}\colon \mathbb{N} \to \mathbb{Z}$ for a function that bounds the total cost of a polynomial product algorithm in terms of the number of ring operations performed independently of

the coefficient ring, assuming a unity is available. In other words, two polynomials of degrees at most d over such a ring $\mathbb{A}$ can be multiplied with at most $\mathsf{M}(d)$ arithmetic operations in $\mathbb{A}$. The schoolbook algorithm allows us to take $\mathsf{M}(d) = O(d^2)$. On the other hand the fastest known algorithm, due to Cantor and Kaltofen [7], provides us with $\mathsf{M}(d) = O(d \log d \log\log d)$. In order to simplify the cost analysis of our algorithms we make the customary assumption that $\mathsf{M}(d_1)/d_1 \leqslant \mathsf{M}(d_2)/d_2$ for all $0 < d_1 \leqslant d_2$. Notice that this assumption implies the *super-additivity* of M, namely $\mathsf{M}(d_1) + \mathsf{M}(d_2) \leqslant \mathsf{M}(d_1 + d_2)$ for all $d_1 \geqslant 0$ and $d_2 \geqslant 0$.

For operations in $\mathbb{Z}$ and in finite fields, we are interested in the *Turing machine model*, with a sufficiently large number of tapes. In short, we use the terms *bit-cost* and *bit-complexity* to refer to this model whenever the context is not clear. For *randomized algorithms*, we endow Turing machines with an additional instruction which generates random bits with a uniform distribution [28].

We write $\mathsf{I}(n)$ for a function that bounds the bit-cost of an algorithm which multiplies two integers of bit-sizes at most n, viewed in classical binary representation. Recently, the best bound for $\mathsf{I}(n)$ has been improved to $\mathsf{I}(n) = O(n \log n\, 8^{\log^* n})$, where $\log^*$ represents the *iterated logarithm* function [16]. Again, we make the customary assumption that $\mathsf{I}(n_1)/n_1 \leqslant \mathsf{I}(n_2)/n_2$ for all $0 < n_1 \leqslant n_2$. We freely use the following classical facts: ring operations in $\mathbb{F}_p$ cost $O(\mathsf{I}(\log p))$ and one division or inversion in $\mathbb{F}_p$ costs $O(\mathsf{I}(\log p) \log\log p)$ [11, Chapter 11].

For polynomial operations over $\mathbb{F}_q$, we let $\mathsf{M}_q(d)$ represent a function that bounds the bit-cost of an algorithm that multiplies two polynomials of degrees at most d, with the same kind of assumptions as for M and I. According to [17], we may take $\mathsf{M}_q(d) = O(d \log q \log(d \log q)\, 8^{\log^*(d \log q)})$. The ring operations in $\mathbb{F}_q$ cost at most $O(\mathsf{M}_p(k))$, and inversions take at most $O(\mathsf{M}_p(k) \log k + \mathsf{I}(\log p) \log\log p)$. For convenience, m_q and d_q will respectively denote cost functions for the product and the inverse in $\mathbb{F}_q$.

Let us recall that the gcd of two polynomials of degrees at most d over $\mathbb{F}_q$ can be computed in time $O(\mathsf{M}_q(d) \log d)$: One can for instance use pseudo-remainders in [11, Algorithm 11.4], *mutatis mutandis*. Given monic polynomials f and g_1, ..., g_l with $\deg f = d$ and $\deg g_1 + \cdots + \deg g_l = O(d)$, all the remainders $f \operatorname{rem} g_i$ can be computed in time $O(\mathsf{M}_q(d) \log l)$ using a *subproduct tree* [11, Chapter 10]. The inverse problem, called *Chinese remaindering*, can be solved within a similar cost $O(\mathsf{M}_q(d) \log l + d\,\mathsf{d}_q)$.

In this article, when needed, we consider that the factorization of $q-1$ and a primitive element of $\mathbb{F}_q^*$ have been precomputed once, and we discard the necessary underlying costs. In practice, if the factorization of $q-1$ is given, then it is straightforward to verify whether a given element is primitive. For known complexity bounds and historical details on these tasks, we refer the reader to [1, 26, 31].

1.3 Related work and our contributions

Seminal algorithms for polynomial factorization over finite fields are classically attributed to Berlekamp [2, 3], and Cantor and Zassenhaus [8], but central earlier ideas can be found in works of Gauss, Galois, Arwins, Faddeev and Skopin. Cantor–Zassenhaus' algorithm is randomized and well suited to compute roots of polynomials of degree d that split over $\mathbb{F}_q$ in average time $O(\mathsf{M}_q(d)\,(\log q + \log d) \log d + d\,\mathsf{d}_q)$. Of course, if $q = O(d)$ then an exhaustive search can be naively performed in time $O(\mathsf{M}_q(d) \log d)$ (the factor $\log d$ can be discarded if a primitive element of $\mathbb{F}_q^*$ is given, by means of [6, Proposition 3]), so that the cost of root finding simplifies to $O(\mathsf{M}_q(d) \log q \log d + d\,\mathsf{d}_q)$. This classical approach is for instance implemented in the NTL library written by Shoup [32]. However neither Berlekamp's nor Cantor–Zassenhaus' algorithm seems to benefit of particular prime numbers such as FFT primes. Instead, alternative approaches have been proposed by Moenck [25], von zur Gathen [10], Mignotte and Schnorr [24], and then by Rónyai [29].

In Section 2, for the sake of comparison, we first revisit Cantor–Zassenhaus' approach and propose a practical trick to slightly speed it up. We also briefly recall the complexity bound of Mignotte–Schnorr's algorithm [24].

We then design and analyze fast randomized algorithms based on tangent Graeffe transforms, leading to an important speed-up for FFT primes. The practical efficiency of the new algorithms is discussed on the basis of implementations in the Mathemagix computer algebra system [18], thus revealing that our fastest variant gains an order of magnitude over other existing software.

Let us finally mention that our present randomized algorithms admit deterministic counterparts which are studied in [14].

2. KNOWN ALGORITHMS REVISTED

In this section we revisit efficient root finding algorithms previously described in the literature, and slightly improve their complexity analysis. The first stream, followed by Berlekamp [3], Cantor and Zassenhaus [8], consists in splitting the input polynomial by computing suitable modular exponentiations and gcds. The second stream, followed by Moenck [25], and Mignotte and Schnorr [24], takes advantage of the smoothness of $q-1$.

2.1 Cantor–Zassenhaus' algorithm

In this subsection we suppose that q has the form $q = \chi\,\rho + 1$, with $\chi \geqslant 2$. The following randomized algorithm extends Cantor–Zassenhaus' one, which corresponds to $\chi = 2$, when q is odd. We need a primitive root ξ of unity of order χ.

Randomized algorithm 1.

Input. $f \in \mathbb{F}_q[x]$ of degree $d \geqslant 1$, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$; a primitive root ξ of unity of order χ.

Output. The roots of f.

1. If $d = O(\chi)$ then compute the roots of f by calling a fallback root finding algorithm.
2. Pick up $g \in \mathbb{F}_q[x]$ at random of degree at most $d-1$.
3. Compute $h := g^\rho \operatorname{rem} f$, and set $f_0 := f$.
4. For all i from 1 to $\chi - 1$, compute $f_i := \gcd(h - \xi^i, f_0)$, make it monic, and replace f_0 by f_0 / f_i.
5. Recursively call the algorithm with those of $f_0, \ldots, f_{\chi-1}$ which are not constant, and return the union of the sets of their roots.

In the classical case when $\chi = 2$, then we recall that step 3 costs $O(\mathsf{M}_q(d)\log q)$ and step 4 takes $O(\mathsf{M}_q(d)\log d)$ plus one inversion in $\mathbb{F}_q$. Since the average depth of the recursion is in $O(\log d)$ [9], the total average cost amounts to $O(\mathsf{M}_q(d)(\log q + \log d)\log d + d\,\mathsf{d}_q)$.

For the case of arbitrary χ, we propose an informal discussion, so as to justify the interest in the algorithm. If α is a root of f, then $h(\alpha) = g(\alpha)^\rho$ has order dividing χ, hence is a power of ξ, or zero with a very small probability. Therefore we have $f = f_0\, f_1 \cdots f_{\chi-1}$. Since g is taken at random, we might expect that the values of $h(\alpha)$ are distributed uniformly among the powers of ξ (we discard cases when $h(\alpha) = 0$). In other words, the depth of the recursive calls is expected to be in $O(\log d / \log \chi)$.

If we further assume that $\log \chi = o(\log q)$, then step 3 takes approximately $C_1\,\mathsf{M}_q(d)\log q$ for a certain constant C_1, and step 4 costs $C_2\ \chi\ \mathsf{M}_q(d)\ \log d$. Discarding the cost of the inversions, and compared to the case $\chi = 2$, we achieve an approximate speed-up of

$$\frac{(C_1 \log q + 2\,C_2 \log d)\,\mathsf{M}_q(d)\,(\log d)/\log 2}{(C_1 \log q + \chi\, C_2 \log d)\,\mathsf{M}_q(d)\,(\log d)/\log \chi}$$

$$= \frac{\log \chi}{\log 2}\,\frac{C_1 \log q/\log d + 2\,C_2}{C_1 \log q/\log d + \chi\, C_2}.$$

Whenever $C_1\ \log\ q\ /\ \log\ d \gg \chi\ C_2$, this speed-up is of order $\log\ \chi\ /\ \log\ 2$. In general, the speed-up is maximal if $\chi(\log \chi - 1) = \frac{C_1 \log q}{C_2 \log d}$.

2.2 Mignotte–Schnorr's algorithm

Assuming given a primitive element of $\mathbb{F}_q^*$, Mignotte and Schnorr proposed a general deterministic root finding algorithm in [24], that is efficient when $q - 1$ is smooth. For a fair comparison with our new algorithm presented in the next section (and since their article is written in French) we recall their method in a different and concise manner.

Let $\pi_1, ..., \pi_m$ be integers $\geqslant 2$, such that $q - 1 = \rho\,\pi_1 \cdots \pi_m$, and let $\chi = \pi_1 \cdots \pi_m$ and $\sigma = \pi_1 + \cdots + \pi_m$. For instance, if the irreducible factorization of $q - 1 = p_1^{m_1} \cdots p_r^{m_r}$ is known, then we may take $\rho = 1$ and $\pi_1 = \pi_2 = \cdots = \pi_{m_1} = p_1$, $\pi_{m_1+1} = \pi_{m_1+2} = \cdots = \pi_{m_1+m_2} = p_2$, ..., and set $m = m_1 + \cdots + m_r$. In order to split f into factors of degrees at most ρ, we may use the following algorithm.

Algorithm 2.

Input. $f \in \mathbb{F}_q[x]$ of degree $d \geqslant 1$, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$; a primitive root ξ of unity of order χ.

Output. (f_1, e_1), ..., (f_s, e_s) in $(\mathbb{F}_q[x] \setminus \mathbb{F}_q) \times \{0, ..., \chi - 1\}$ such that $f = f_1 \cdots f_s$, and the f_i are monic, separable, and divide $x^\rho - \xi^{e_i}$, for all i.

1. Let $h_0(x) := x^\rho \operatorname{rem} f(x)$, and
   compute $h_i(x) := h_{i-1}(x)^{\pi_i} \operatorname{rem} f(x)$, for all $1 \leqslant i \leqslant m-1$.
2. Initialize F with the list $[(f, 0)]$.
3. For i from m down to 1 do
   a. Compute $h_{i-1} \operatorname{rem} g$ for all pairs (g, e) in F using a subproduct tree.
   b. Initialize G with the empty list, and for all j from 0 to $\pi_i - 1$ do
      For each pair (g, e) in F do
         Compute $g_j := \gcd((h_{i-1} \operatorname{rem} g) - \xi^{(e+j\chi)/\pi_i}, g)$.
         If g_j is not constant, then make it monic and append $(g_j, (e + j\chi)/\pi_i)$ to G.
      Let $F := G$.
4. Return F.

Lemma 3. *Algorithm 2 is correct and executes in time* $O(\sigma\,\mathsf{M}_q(d)\log d + \mathsf{M}_q(d)\log q + m\,d\,\mathsf{I}(\log \chi) + m\,\mathsf{m}_q\,d\log \chi + d\,\mathsf{d}_q) = \sigma\,\tilde{O}(d \log q) + \tilde{O}(d \log^3 q)$.

Proof. We prove the correctness by descending induction on i from m down to 1. In fact, at the end of iteration i of the loop of step 3, we claim that $f = \prod_{(g,e)\in F} g$, that g divides $x^{\rho\pi_1\cdots\pi_{i-1}} - \xi^e$ for all $(g,\ e) \in F$, and that the integers e in F are divisible by $\pi_1 \cdots \pi_{i-1}$ and bounded by $\chi - 1$. By construction, these properties are all true with $i = m + 1$ when entering step 3, so that by induction we can assume that these properties hold for $i + 1$ when entering the loop at level i. Let $(g, e) \in F$, and let α be a root of g. From $\alpha^{\rho\pi_1\cdots\pi_i} = \xi^e$, since e is divisible by π_i, there exists $j \in \{0, ..., \pi_i - 1\}$ such that $\alpha^{\rho\pi_1\cdots\pi_{i-1}} = \xi^{\frac{e+j\chi}{\pi_i}}$. We are done with the correctness.

As to the complexity, we can compute the χ/π_i for $i \in \{1, ..., m\}$ in time $O(m\,\mathsf{I}(\log \chi))$ as follows: we first compute π_1, $\pi_1\,\pi_2$, $\pi_1\,\pi_2\,\pi_3$, ..., $\pi_1\,\pi_2 \cdots \pi_{m-1}$, and π_m, $\pi_{m-1}\,\pi_m$, $\pi_{m-2}\,\pi_{m-1}\,\pi_m$, ..., $\pi_2 \cdots \pi_{m-1}\,\pi_m$, and then deduce each χ/π_i by multiplying $\pi_1 \cdots \pi_{i-1}$ and $\pi_{i+1} \cdots \pi_m$.

Step 1 requires time $O(\mathsf{M}_q(d)\log q)$. In step 3.a, since the sum of the degrees of the polynomials in F is at most d, and since these polynomials are monic, this step can be done in time $O(\mathsf{M}_q(d)\log d)$. The cost of the gcds in step 3.b is $O(\pi_i\,\mathsf{M}_q(d)\log d)$ by the super-additivity of M_q. We compute all the e/π_i in time $O(d\,\mathsf{I}(\log \chi))$, and then all the ξ^{e/π_i} and ξ^{χ/π_i} by means of $O(d \log \chi)$ operations in $\mathbb{F}_q$. Deducing all the $\xi^{(e+j\chi)/\pi_i}$ takes $O(\pi_i\,d)$ additional products in $\mathbb{F}_q$.

Since the cardinality of F cannot exceed d, the total number of proper factors g_j in step 3.b is at most d. Therefore, we need $O(d)$ inversions in order to make all the polynomials in F monic. □

Notice that the dependence on σ is good when $q - 1$ is smooth. For example, if $\rho = 1$ and $\sigma = O(\log q)$, then the cost of root finding *via* the latter proposition drops to $\tilde{O}(d \log^3 q)$. This is higher than the average $\tilde{O}(d \log^2 q)$ bit-cost of Cantor–Zassenhaus' algorithm. Nevertheless since the term $\tilde{O}(d\ \log^3\ q)$ corresponds to $O(d\ \log^2\ q)$ products in $\mathbb{F}_q$, with a small constant hidden in the O, Mignotte–Schnorr's algorithm is competitive for small values of $\log q$.

In the original work of Mignotte and Schnorr [24], the cost of the algorithm was estimated to $O(\mathsf{M}(d)\sum_{i=1}^r m_i(\log q + p_i\ \log\ d))$ operations in $\mathbb{F}_q$, if $q\ -\ 1\ =\ p_1^{m_1}\ \cdots\ p_r^{m_r}$. Our presentation slightly differs by the use of a subproduct tree in step 3.a.

Let us briefly mention that a better bound for splitting f was achieved in [10]. Nevertheless, for finding all the roots, the method of [10] does not seem competitive in general.

2.3 Moenck's algorithm

Moenck's algorithm [25] deals with the special case when $q-1=\rho\,2^m$. Let ζ be a primitive root of $\mathbb{F}_q^*$. Let $1\leqslant i\leqslant m$, let f be a polynomial whose roots have orders dividing $\rho\,2^i$, and let $\alpha\in\mathbb{F}_q^*$ be one of these roots. Then either this order divides $\rho\,2^{i-1}$, or we have $\alpha^{\rho 2^{i-1}}+1=0$, since $x^{\rho 2^i}-1=(x^{\rho 2^{i-1}}+1)\,(x^{\rho 2^{i-1}}-1)$. In the latter case, we obtain $(\alpha/\zeta^{2^{m-i}})^{\rho 2^{i-1}}=1$. The polynomials $f_1=\gcd(x^{\rho 2^{i-1}}-1,f)$ and $f_2(x)=(f/f_1)(\zeta^{2^{m-i}}x)$ have all their roots of order dividing $\rho\,2^{i-1}$. In this way, the roots of f can be computed inductively starting from $i=m$ down to $i=1$. At the end, we are reduced to finding roots of several polynomials whose orders divide ρ. If ρ is small, then an exhaustive search easily completes the computations. Otherwise we may use a fallback algorithm. Moenck's algorithm summarizes as follows.

Algorithm 4.

Input. $f\in\mathbb{F}_q[x]$ of degree $d\geqslant 1$, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0)\neq 0$; a primitive element ζ of $\mathbb{F}_q^*$.

Output. $(f_1,\ \gamma_1)$, ..., $(f_s,\ \gamma_s)$ in $(\mathbb{F}_q[x]\setminus\mathbb{F}_q)\times\mathbb{F}_q$ such that $f=f_1\cdots f_s$, the f_i are monic, separable, and the roots of $f_i(\gamma_i\,x)$ have orders dividing ρ.

1. Compute $h_i(x):=x^{\rho 2^i}\operatorname{rem} f(x)$ for all $i\in\{0,...,m-1\}$.
2. Initialize F with the list $[(f,1)]$.
3. For i from m down to 1 do
   a. Compute the remainders of $h_{i-1}\operatorname{rem} g$ for all triples (g,γ) in F using a subproduct tree.
   b. Initialize G as an empty list.
   c. For all (g,γ) in F do
      i. Compute $g_1(x):=\gcd((h_{i-1}\operatorname{rem} g)(x)-\gamma^{\rho 2^{i-1}},\ g(x))$. If g_1 is not constant then make it monic and append (g_1,γ) to G.
      ii. Compute $g_2:=g/g_1$. If g_2 is not constant then make it monic, and append $(g_2,\gamma\zeta^{2^{m-i}})$ to G.
   d. Replace F by G.
4. Return F.

Our presentation differs from [25]. We also introduced step 3.a, which yields a sensible theoretical speed-up, similarly to Mignotte–Schnorr's algorithm. In fact Moenck's and Mignotte–Schnorr's algorithms are quite similar from the point of view of the successive splittings of f, and the intermediate operations. We thus leave out the details on correctness and cost analysis. As an advantage, the logarithms of the successive roots are not needed. Nevertheless computing all the powers of γ in steps 3.c amounts to a bit-cost $\tilde{O}(d\log^3 q)$.

3. USING GRAEFFE TRANSFORMS

One advantage of Cantor–Zassenhaus' algorithm is its average depth in the recursive calls in $O(\log d)$. This is to be compared to the $O(\log q)$ iterations of Algorithms 2 and 4. In this section we propose a new kind of root finder based on the tangent Graeffe transform, which takes advantage of a FFT prime field, with an average cost not exceeding the one of Cantor–Zassenhaus.

3.1 Generalized Graeffe transforms

Classically, the *Graeffe transform* of a polynomial $g\in\mathbb{F}_q[x]$ of degree d is the unique polynomial $h\in\mathbb{F}_q[x]$ satisfying $h(x^2)=g(x)\,g(-x)$. If $g(x)=\prod_{i=1}^d(\alpha_i-x)$, then $h(x)=\prod_{i=1}^d(\alpha_i^2-x)$. This construction can be extended to higher orders as follows: the *generalized Graeffe transform* of g of order π, written $G_\pi(g)$, is defined as the resultant

$$G_\pi(g)(x)=(-1)^{\pi d}\operatorname{Res}_z(g(z),z^\pi-x).$$

If $g=\prod_{i=1}^d(\alpha_i-x)$, then $G_\pi(g)(x)=\prod_{i=1}^d(\alpha_i^\pi-x)$. Equivalently, $G_\pi(g)$ is the characteristic polynomial of multiplication by x^π in $\mathbb{F}_q[x]/(g)$ (up to the sign). For our root finding algorithms, the most important case is when $q-1$ is smooth and the order π of the generalized Graeffe transform divides $q-1$.

Proposition 5. *Let $\pi_1,...,\pi_m$ be integers $\geqslant 2$, such that $\chi=\pi_1\cdots\pi_m$ divides $q-1$, and let ξ_i be given primitive roots of unity of orders π_i, for all $i\in\{1,...,m\}$. If g is a monic polynomial in $\mathbb{F}_q[x]$ of degree d, then the generalized Graeffe transforms of orders $\pi_1,\pi_1\pi_2,\pi_1\pi_2\pi_3,...,\chi$ of g can be computed in time $O(m\,\mathsf{M}_q(\mu\,d))$ or $O(\mathsf{M}_q(\sigma\,d))$, where $\mu=\max(\pi_1,...,\pi_m)$ and $\sigma=\pi_1+\cdots+\pi_m$.*

Proof. Writing $g(x)=c\prod_{j=1}^d(\alpha_j-x)$ in an algebraic closure of $\mathbb{F}_q$, the Graeffe transform of g of order π_i is $h_i(x)=c^{\pi_i}\prod_{j=1}^d(\alpha_j^{\pi_i}-x)$. Consequently this leads to $h_i(x^{\pi_i})=g(x)\,g(\xi_i\,x)\,g(\xi_i^2\,x)\cdots g(\xi_i^{\pi_i-1}\,x)$. Using the latter formula, by Lemma 6 below, the transform can be obtained in time $O(\mathsf{M}_q(\pi_i\,d))$. Taking the sum over i concludes the proof. □

Lemma 6. *Let g be a polynomial of degree $d\geqslant 1$ in $\mathbb{F}_q[x]$, let $\alpha\in\mathbb{F}_q$, and let l be an integer. Then the product $P_l(x)=g(x)\,g(\alpha\,x)\,g(\alpha^2\,x)\cdots g(\alpha^{l-1}\,x)$ can be computed in time $O(\mathsf{M}_q(l\,d))$.*

Proof. Let $h:=\lfloor l/2\rfloor$. If l is even, then we have $P_l(x)=P_h(x)\,P_h(\alpha^h\,x)$, otherwise we have $P_l(x)=P_h(x)\,P_h(\alpha^h\,x)\,g(\alpha^{l-1}\,x)$. These formulas lead to an algorithm with the claimed cost. □

Let us mention that good complexity bounds for Graeffe transforms for general finite fields can be found in [14].

3.2 Tangent Graeffe transforms

Introducing a formal parameter ε with $\varepsilon^2=0$, we define the *generalized tangent Graeffe transform* of g of order π as being $G_\pi(g(x+\varepsilon))\in(\mathbb{F}_q[\varepsilon]/(\varepsilon^2))[x]$. For any ring R, computations with "tangent numbers" in $R[\varepsilon]/(\varepsilon^2)$ can be done with constant overhead with respect to computations in R (in the FFT model, the overhead is asymptotically limited to a factor of two). Whenever a Graeffe transform preserves the number of distinct simple roots, the tangent Graeffe transform can be used to directly recover the original simple roots from the transformed polynomial, as follows:

Lemma 7. *Let $g\in\mathbb{F}_q[x]$ be separable of degree d, let π be coprime to p, and let $h(x)+\bar{h}(x)\,\varepsilon+O(\varepsilon^2)=G_\pi(g(x+\varepsilon))$. A nonzero root β of h is simple if, and only if, $\bar{h}(\beta)\neq 0$. For such a root, $\pi\,\beta\,h'(\beta)/\bar{h}(\beta)$ is the unique root α of g such that $\alpha^\pi=\beta$.*

PROOF. Let α be a root of g in an algebraic closure of $\mathbb{F}_q$. The lemma follows from the formula $\bar{h}(\alpha^\pi) = \pi\,\alpha^{\pi-1}\,h'(\alpha^\pi)$, obtained by direct calculation. □

In the context of the Graeffe method, the tangent transform is classical (for instance, see [23, 27] for history, references, and use in numerical algorithms). The generalized tangent Graeffe transform can also be seen as the *tangent characteristic polynomial* of x^π modulo $g(x+\varepsilon)$, and this construction is often attributed to Kronecker in algebraic geometry [22].

3.3 Overview of our methods

Let us write $q-1 = \rho\,\pi_1 \cdots \pi_m$. Let $h_0 := f$, and let h_1, ..., h_m be the Graeffe transforms of orders π_1, $\pi_1\,\pi_2$, ..., $\pi_1 \cdots \pi_m$ of f. The roots of h_m are to be found among the elements of order ρ. One may then find the roots of h_{m-1} among the π_m-th roots of the roots of h_m, and, by induction, the roots of h_{i-1} are to be found among the π_i-th roots of the roots of h_i. This is the starting point of the efficient deterministic algorithm designed in [14]. But in order to make it much more efficient than Cantor–Zassenhaus's algorithm we introduce and analyze two types of randomizations in the next sections. First we avoid computing π_i-th roots by combining random variable shifts and tangent transforms. Second we analyze the behaviour for random polynomials, which leads us to a very efficient heuristic algorithm.

3.4 A randomized algorithm

Our randomized algorithms can be competitive to Cantor–Zassenhaus' algorithm only for very smooth values of $q-1$. For simplicity we focus on the important case of an FFT field where $q = M\,2^m + 1$, with $M = O(\log q)$.

We introduce the parameter $\tau \in \mathbb{F}_q$. Let $\chi = M\,2^{m-l}$ be a divisor of $q-1$, and let $\rho = (q-1)/\chi = 2^l$. Let $\alpha_1, ..., \alpha_d$ denote the roots of f. The Graeffe transform g_τ of $f(x-\tau)$ of order ρ equals $\prod_{i=1}^{d} ((\alpha_i+\tau)^\rho - x)$.

Given a pair of distinct roots α_i and α_j of f, we have $(\tau+\alpha_i)^\rho \neq (\tau+\alpha_j)^\rho$ for all but at most ρ values of τ. Therefore for all but at most $\frac{d\,(d-1)}{2}\,\rho$ values of τ, the polynomial g_τ has no multiple root. Considering that τ is taken uniformly at random in $\mathbb{F}_q$, the probability that g_τ has multiple roots is at most $\frac{d\,(d-1)}{2}\,\frac{1}{\chi}$. This yields the following randomized algorithm.

Randomized algorithm 8.

Input. $f \in \mathbb{F}_q[x]$ of degree $d \geqslant 1$, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$; a primitive root ζ of $\mathbb{F}_q^*$.

Output. The roots of f.

1. If $d\,(d-1) \geqslant q$ then set l to 0. Otherwise set l to the largest integer in $\{0, ..., m\}$ such that $d\,(d-1) \leqslant M\,2^{m-l}$. Compute $\chi := M\,2^{m-l}$ and $\rho := (q-1)/\chi = 2^l$.
2. Pick $\tau \in \mathbb{F}_q$ uniformly at random, and compute $h_0(x) + \varepsilon\,\bar{h}_0(x) := f(x-\tau+\varepsilon)$, where $\varepsilon^2 = 0$.
3. Recursively compute the Graeffe transform $h_i + \varepsilon\,\bar{h}_i$ of order 2 of $h_{i-1} + \varepsilon\,\bar{h}_{i-1}$, for all $1 \leqslant i \leqslant m$.
4. Compute the list $E := [j\ 2^m |\ j \in \{0, ..., M-1\}, h_m(\zeta^{j\,2^m}) = 0]$ of ζ-logarithms of the roots of h_m.
5. For i from m down to $l+1$ do
   a. Replace E by the concatenation of $[(e + j\,\chi)/2 |\ e \in E]$ for $j \in \{0, 1\}$.
   b. If E has cardinality more than d then remove the elements e from E such that $h_{i-1}(\zeta^e) \neq 0$.
6. If $l = 0$ then return $\{\zeta^e - \tau |\ e \in E\}$.
7. Compute $E_1 := [e |\ e \in E, \bar{h}_l(\zeta^e) \neq 0]$.
8. Compute $Z_1 := [2^l\,\zeta^e\,h_l'(\zeta^e)/\bar{h}_l(\zeta^e) - \tau |\ e \in E_1]$. Add τ to Z_1 if $f(\tau) = 0$.
9. Compute $f_2(x) := f(x)/\prod_{a \in Z_1} (x-a)$.
10. Compute recursively the roots Z_2 of f_2.
11. Return $Z_1 \cup Z_2$.

PROPOSITION 9. *Algorithm 8 is correct, and takes an average time $\tilde{O}(d\,\log^2 q)$, if $d < q$ and $q = M\,2^m + 1$ with $M = O(\log q)$.*

PROOF. The correctness follows from Lemma 7, which asserts that Z_1 is a subset of the roots of f.

Step 3 takes $\tilde{O}(d\,\log^2 q)$ by Proposition 5. Steps 2, 5.a, 5.b, 6, 7, 8, 9 execute in time $\tilde{O}(d\,\log^2 q)$. Step 4 costs $\tilde{O}((d+M)\,\log q) = \tilde{O}(d\,\log q + \log^2 q)$. If $d\,(d-1) \geqslant q$ then the number of iterations in step 5 is $m = O(\log q) = O(\log d)$. Otherwise the number of iterations is $m - l = O(\log d)$. Consequently, the total cost of all steps but step 10 is $\tilde{O}(d\,\log^2 q)$. From the choice of l, we have already seen that the degree of f_2 equals 0 with probability at least $1/2$. Writing $T(d)$ for the average execution time, we thus have $T(d) \leqslant \tilde{O}(d\,\log^2 q) + \frac{1}{2}\,T(d)$, which implies $T(d) = \tilde{O}(d \log^2 q)$. □

Since the base field supports FFTs it is important to perform all intermediate computations by taking advantage of sharing transforms. This technique was popularized within the NTL library for many algorithms. In addition we notice that the first few iterates of the loop of step 5 can be reduced to direct FFT computations instead of generic multi-point evaluations.

3.5 Random polynomials which split over $\mathbb{F}_q$

Taking a monic random polynomial of degree d which splits over $\mathbb{F}_q$, for a uniform distribution, is equivalent to taking its d roots at random in $\mathbb{F}_q$. Let $\alpha_1, ..., \alpha_d$ represent the roots of f. We examine the behavior of tangent Graeffe transforms of such random polynomials.

Let $P_{d,i}$ be the number of monic polynomials of degree d which split over $\mathbb{F}_q$ and have i distinct nonzero roots of order dividing χ. Such a polynomial uniquely corresponds to the choice of i distinct values among χ, namely the roots, and then of the multiplicities of these roots with sum d. Therefore we have $P_{d,i} = \binom{\chi}{i}\binom{d-1}{i-1}$. One can check that $\sum_{i=0}^{d} P_{d,i} = \binom{\chi+d-1}{\chi-1}$, which corresponds to choosing a multiplicity in $\{0, ..., d\}$ for each element of order dividing χ, under the constraint that the sum of the multiplicities is exactly d. The average number of roots of such polynomials is

$$\frac{\sum_{i=0}^{d} i\,P_{d,i}}{\sum_{i=0}^{d} P_{d,i}} = \frac{\chi\binom{\chi+d-2}{\chi-1}}{\binom{\chi+d-1}{\chi-1}} = \frac{d}{1+(d-1)/\chi}.$$

The latter formulas are direct consequences of the classical Chu–Vandermonde identity. Assuming that $\chi \geqslant d$, the average number of distinct roots is at least $d/2$. When χ is much greater than d then this lower bound is getting close to d.

PROPOSITION 10. *Let ρ be a fixed divisor of $q-1$, and let $\chi = (q-1)/\rho$. Let $f \in \mathbb{F}_q[x]$ be a monic separable polynomial of degree $d \leqslant q-1$, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$. Let g be the Graeffe transform of f of order ρ. The average number of simple roots of g over all such polynomials f endowed with a uniform distribution is at least $d\frac{1-d/\chi}{1+d/\chi}$.*

Moreover, assuming that $\chi > 3\,d$ the probability that g has at least $\left(2-\frac{1+d/\chi}{1-d/\chi}\right)d$ simple roots is at least $1/2$.

PROOF. The polynomial g is uniformly distributed in the set of monic polynomials with roots of order χ. Let s and m be the respective numbers of simple and multiple roots of g, so that $d \geqslant s + 2\,m$. We have just seen that the average number $s+m$ of distinct roots of g is at least $\frac{d}{1+d/\chi}$. Hence, $s+(d-s)/2 \geqslant \frac{d}{1+d/\chi}$ and $s \geqslant \frac{2\,d}{1+d/\chi} - d = d\frac{1-d/\chi}{1+d/\chi}$.

Setting $K = \frac{1-d/\chi}{1+d/\chi}$ and assuming that $\chi > 3\,d$, so that $2 > 1/K > 1$, let P be the probability that g has less than $(2-K)\,d$ simple roots. Then the average number of roots $K\,d$ is at most $P\,(2-1/K)\,d + (1-P)\,d$, whence $(K-1)\,d \leqslant P\,(1-1/K)\,d$. We conclude that $P \geqslant (K-1)/(1-1/K) = K > 1/2$. □

3.6 A heuristic randomized algorithm

Let χ be a divisor of $q-1$, $\rho = (q-1)/\chi$ and consider a polynomial $f \in \mathbb{F}_q[x]$ of degree d. In Section 3.4, we have shown that for all but at most $\frac{d\,(d-1)}{2}\,\rho$ values of τ, the Graeffe transform g_τ of order ρ of $f(x-\tau)$ has no multiple root. Taking $\chi > d^2$, this implies that g_τ has no multiple roots with probability at least $1/2$, when picking τ at random. Now Proposition 10 implies that at least one third of the roots remain simple with probability at least $1/2$ under the weaker condition that $\chi > 4\,d$ and for random f. It is an interesting question whether this fact actually holds for any f if τ is randomly chosen:

HEURISTIC. *Let $q-1 = \rho\,\chi$ with $\chi > 4\,d$ and let $f \in \mathbb{F}_q[x]$ be a monic, separable polynomial of degree d which splits over $\mathbb{F}_q$. Then there exist at least $q/2$ elements $\tau \in \mathbb{F}_q[x]$ such that the Graeffe transform of order ρ of $f(x-\tau)$ has at least $d/3$ simple roots.*

From now on, assume that f is picked at random or that the above heuristic holds and τ is chosen at random. Taking $\chi > 4\,d$, we may then find the roots of g_τ by computing a few discrete Fourier transforms of length $O(d)$, instead of more expensive multi-point evaluations. Using tangent Graeffe transforms, the simple roots can be lifted back for almost no cost. These considerations lead to the following efficient randomized algorithm.

Randomized algorithm 11.

Input. $f \in \mathbb{F}_q[x]$ of degree $d \geqslant 1$, monic, separable, which splits over $\mathbb{F}_q$, and such that $f(0) \neq 0$; a primitive element ζ of $\mathbb{F}_q^*$.

Output. The roots of f.

1. If $d \geqslant M\,2^{m-3}$ then evaluate f on all the elements of $\mathbb{F}_q^*$, and return the roots.
2. Otherwise set l to be the largest integer in $\{1, ..., m-2\}$ such that $d < M\,2^{m-l-2}$. Compute $\chi := M\,2^{m-l}$ and $\rho := (q-1)/\chi = 2^l$.
3. Pick a random τ in $\mathbb{F}_q$, and compute the Graeffe transform $h_l(x) + \varepsilon\,\bar{h}_l(x)$ of order ρ of $f(x-\tau+\varepsilon)$ with $\varepsilon^2 = 0$.
4. Compute $[h_l(\zeta^{j\,2^l}) |\ j \in \{0, ..., \chi-1\}]$, $[h_l'(\zeta^{j\,2^l}) |\ j \in \{0, ..., \chi-1\}]$, $[\bar{h}_l(\zeta^{j\,2^l}) |\ j \in \{0, ..., \chi-1\}]$.
5. Compute $Z_1 := \left[2^l\,\zeta^{j\,2^l}\,h_l'\left(\zeta^{j\,2^l}\right)/\bar{h}_l\left(\zeta^{j\,2^l}\right) - \tau \,|\, j \in \{0, ..., \chi-1\}, h_l(\zeta^{j\,2^l}) = 0, \bar{h}_l(\zeta^{j\,2^l}) \neq 0\right]$. Add τ to Z_1 if $f(\tau) = 0$.
6. Compute $f_2(x) := f(x)/\prod_{a \in Z_1}(x-a)$.
7. Compute recursively the roots Z_2 of f_2.
8. Return $Z_1 \cup Z_2$.

PROPOSITION 12. *Assume the heuristic or that f is chosen at random among the monic, separable polynomials with $f(0) \neq 0$ and which split over $\mathbb{F}_q$. Then Algorithm 11 takes an average time $O(\mathsf{M}_q(d)\,\log\,q) = \tilde{O}(d\,\log^2\,q)$, whenever $d < q$ and $q = M\,2^m + 1$ with $M = O(\log q)$.*

PROOF. Of course if $d > M\,2^{m-3}$ then the cost is $O(\mathsf{M}_q(d)\log d)$. Now suppose we arrive in step 2. The roots of h_l have order dividing χ, and we have $\chi/2 \leqslant 4\,d < \chi$. If the algorithm terminates then it is correct by the same arguments as for Algorithm 8.

The cost of step 3 is bounded by $O(\mathsf{M}_q(d)\,\log\,d)$ (and even by $O(\mathsf{M}_q(d))$ if $p > d$ according to [4, Chapter 1, Section 2]). When computing h_l using l tangent Graeffe transforms of order 2, the cost of step 3 is bounded by $O(\mathsf{M}_q(d)\,l) = O(\mathsf{M}_q(d)\,\log\,q)$. Steps 4 and 5 can be done in time $O(\mathsf{M}_q(d)\,\log\,d)$. The average execution time $T(d)$ thus satisfies

$$T(d) \;\leqslant\; O(\mathsf{M}_q(d)\log q) + PT(d) + (1-P)\,T(2\,d/3),$$

where $P \leqslant 1/2$ is the probability that Z_2 contains more than $2\,d/3$ elements in step 7. It follows that $T(d) = O(\mathsf{M}_q(d)\,\log\,q) + T(2\,d/3)$, whence $T(d) = O(\mathsf{M}_q(d)\log q)$. □

The main bottleneck for Algorithm 11 is step 3. It would be possible to improve our complexity bound if a better algorithm were known to compute Graeffe transforms of order 2^l. In the FFT model, the tangent Graeffe transform of a polynomial of degree d can be computed in time $(4/3 + o(1))\;\mathsf{M}_q(d)$. In practice, this means that the asymptotic complexity of Algorithm 11 is equivalent to $4/3\,\mathsf{M}_q(d)\log_2 q$.

In order to reduce the cost of the Graeffe transforms, it is relevant to choose l such that $M\,2^{m-l-\eta-1} < d \leqslant M\,2^{m-l-\eta}$ for some small $\eta \geqslant 2$, which is to be determined in practice as a function of d and q. In this way more time is spent in multi-point evaluations. For the multi-point evaluations in step 4, one may use M FFTs of size 2^{m-l} for small values of M, or appeal to Bluestein's chirp transform [5].

4. TIMINGS

In this section we report on timings for the algorithms presented above. We use a platform equipped with an Intel® Core™ *i7-4770* CPU at 3.40 GHz, and 8 GB of *1600 MHz DDR3*. It runs the Jessie GNU Debian® operating system with a Linux® kernel version 3.14 in 64 bit mode. We compile with GCC [12] version 4.9.1.

We use revision 9738 of Mathemagix. For the sake of comparison to other software, we installed the NTL library version 8.0.0 [32] (configured to benefit from GMP [13] version 6.0.0), and Flint version 2.4.4 [15]. For our benchmark family we simply build polynomials f of degree d from random pairwise distinct roots.

Table 1 shows timings for the FFT prime field $\mathbb{F}_q = \mathbb{F}_p$ with $p = 7 \cdot 2^{26} + 1$. The row NTL corresponds to the function `FindRoots`, which contains an optimization with respect to Algorithm 1 with $\chi = 2$: in fact, the polynomials g in step 2 are taken of degree 1 (see [30] for details). We implemented the same optimization in our versions of Cantor–Zassenhaus in Mathemagix. The row Flint corresponds to the function `nmod_poly_factor_equal_deg`, that does not take advantage of this optimization, which mainly explains a loss of efficiency. The rows "Alg. 1" to "Alg. 11" correspond to our own implementations in Mathemagix of the above algorithms (see files `algebramix/polynomial_modular_int.hpp` and `algebramix/bench/polynomial_modular_int_bench.cpp`). Our modified version of Cantor–Zassenhaus' algorithm, with $\chi > 2$, does not reveal to be of practical interest for this prime size.

Concerning the deterministic algorithms, let us precise that we take $\pi_1 = \cdots = \pi_{m-1} = 2$ and $\pi_m = M$ in Algorithm 2. In addition, in our implementation of Algorithm 4, when g and i are small, for efficiency reasons, we compute $h_{i-1} \operatorname{rem} g$ using binary powering instead of simultaneous remainders. These deterministic methods turn out to be quite competitive.

Our Table 2 is similar to Table 1 for the FFT prime field $\mathbb{F}_q = \mathbb{F}_p$ with $p = 5 \cdot 2^{55} + 1$. Nevertheless NTL does not support this larger prime within its type `zz_p`. This larger prime starts to reveal the interest in our modified version of Cantor–Zassenhaus with $\chi > 2$. The speed-up measured with the modified version even more increases with larger primes: With $p = 7 \cdot 2^{120} + 1$, it reaches a factor 1.6 in size 2^{14}.

Using Graeffe transforms as in Algorithm 8 is not very competitive. However, in final, it is satisfactory to observe that our Algorithm 11, exploiting the tangent transform, gains by an order of magnitude over other existing methods in both tables.

$d+1$	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
NTL	0.0065	0.015	0.037	0.090	0.20	0.48	1.1	2.5	5.4	12	26
Flint	0.0093	0.025	0.065	0.17	0.45	1.2	3.0	8.0	22	63	250
Alg. 1, Cantor–Zassenhaus, $\chi = 2$	0.010	0.020	0.043	0.092	0.20	0.42	0.90	1.9	4.0	8.6	18
Alg. 1, Cantor–Zassenhaus, modified	0.009	0.018	0.036	0.091	0.20	0.41	0.96	2.1	4.4	10	22
Alg. 2, Mignotte–Schnorr	0.063	0.12	0.25	0.51	1.0	2.0	3.9	7.8	15	31	62
Alg. 4, Moenck	0.025	0.051	0.10	0.21	0.44	0.91	1.9	3.9	8.2	17	35
Alg. 8, Graeffe, randomized	0.003	0.006	0.021	0.074	0.25	0.70	1.7	3.6	7.5	15	29
Alg. 11, Graeffe, heuristic	0.001	0.002	0.003	0.007	0.015	0.031	0.065	0.13	0.24	0.59	1.4

Table 1. Randomized root finding in degree d over $\mathbb{F}_p$, with $p = 7 \cdot 2^{26} + 1$, time in seconds.

$d+1$	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
Flint	0.032	0.084	0.22	0.58	1.5	3.8	9.5	23	58	150	470
Alg. 1, Cantor–Zassenhaus, $\chi = 2$	0.035	0.075	0.16	0.37	0.78	1.7	3.8	8.6	19	41	90
Alg. 1, Cantor–Zassenhaus, modified	0.025	0.055	0.11	0.28	0.63	1.3	3.1	7.0	14	33	76
Alg. 2, Mignotte–Schnorr	0.22	0.46	0.96	2.0	4.2	8.7	18	39	83	180	370
Alg. 4, Moenck	0.89	0.18	0.37	0.76	1.6	3.4	7.2	16	33	72	150
Alg. 8, Graeffe, randomized	0.011	0.038	0.14	0.56	1.1	2.6	6.8	16	40	92	210
Alg. 11, Graeffe, heuristic	0.005	0.007	0.016	0.032	0.067	0.14	0.29	0.61	1.3	2.7	5.6

Table 2. Randomized root finding in degree d over $\mathbb{F}_p$, with $p = 5 \cdot 2^{55} + 1$, time in seconds.

Coming back to our initial motivation for sparse polynomial interpolation, let us mention that plugging Algorithm 11 into our implementations reported in [19] leads to quite good speed-ups. The total time for Example 1 of [19], concerning the product of random sparse polynomials, decreases from 12 s to 7.6 s with the "coefficient ratios" technique. In fact, the time spent in root finding decreases from 50 % to 8 % during the sole determination of the support.

Example 2 of [19] concerns the sparse interpolation of the determinant of the generic $n \times n$ matrix. For $n = 8$, the time for the "Kronecker substitution" technique with a prime of 68 bits decreases from 95 s to 43 s. The time elapsed in root finding during the sole determination of the

support drops from 70 % to 20 %. Thanks to additional optimizations, the "coefficient ratios" technique now runs this example in 46 s.

Acknowledgements
Bruno Grenet was partially supported by a LIX–Qualcomm®–Carnot postdoctoral fellowship.

5. REFERENCES

[1] E. Bach. Comments on search procedures for primitive roots. *Math. Comp.*, 66:1719–1727, 1997.

[2] E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Tech. J.*, 46:1853–1859, 1967.

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

[4] D. Bini and V. Y. Pan. *Polynomial and matrix computations. Vol. 1. Fundamental algorithms*. Progress in Theoretical Computer Science. Birkhäuser, 1994.

[5] L. I. Bluestein. A linear filtering approach to the computation of discrete Fourier transform. *IEEE Transactions on Audio and Electroacoustics*, 18(4):451–455, 1970.

[6] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

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

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

[9] Ph. Flajolet and J.-M. Steyaert. A branching process arising in dynamic hashing, trie searching and polynomial factorization. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming. Proceedings of the 9th ICALP Symposium*, volume 140 of *Lecture Notes in Comput. Sci.*, pages 239–251. Springer Berlin Heidelberg, 1982.

[10] J. von zur Gathen. Factoring polynomials and primitive elements for special primes. *Theoret. Comput. Sci.*, 52(1-2):77–89, 1987.

[11] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, 2nd edition, 2003.

[12] GCC, the GNU Compiler Collection. Software available at http://gcc.gnu.org, from 1987.

[13] T. Granlund et al. GMP, the GNU multiple precision arithmetic library, from 1991. Software available at http://gmplib.org.

[14] B. Grenet, J. van der Hoeven, and G. Lecerf. Deterministic root finding over finite fields using Graeffe transforms. Technical report, École polytechnique, 2015.

[15] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2014. Version 2.4.4, http://flintlib.org.

[16] D. Harvey, J. van der Hoeven, and G. Lecerf. Even faster integer multiplication. http://arxiv.org/abs/1407.3360, 2014.

[17] D. Harvey, J. van der Hoeven, and G. Lecerf. Faster polynomial multiplication over finite fields. http://arxiv.org/abs/1407.3361, 2014.

[18] J. van der Hoeven et al. Mathemagix, from 2002. http://www.mathemagix.org.

[19] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation in practice. *ACM Commun. Comput. Algebra*, 48(4), 2014. In section "ISSAC 2014 Software Presentations".

[20] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp.*, 67(223):1179–1197, 1998.

[21] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In A. Z. Broder et al., editors, *49th Annual IEEE Symposium on Foundations of Computer Science 2008 (FOCS '08)*, pages 146–155. IEEE, 2008.

[22] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *J. reine angew. Math.*, 92:1–122, 1882.

[23] G. Malajovich and J. P. Zubelli. Tangent Graeffe iteration. *Numer. Math.*, 89(4):749–782, 2001.

[24] M. Mignotte and C. Schnorr. Calcul déterministe des racines d'un polynôme dans un corps fini. *C. R. Acad. Sci. Paris Sér. I Math.*, 306(12):467–472, 1988.

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

[26] G. L. Mullen and D. Panario. *Handbook of Finite Fields*. Discrete Mathematics and Its Applications. Chapman and Hall/CRC, 2013.

[27] V. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Rev.*, 39(2):187–220, 1997.

[28] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[29] L. Rónyai. Factoring polynomials modulo special primes. *Combinatorica*, 9(2):199–206, 1989.

[30] V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In S. M. Watt, editor, *ISSAC '91: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 14–21. ACM Press, 1991.

[31] V. Shoup. Searching for primitive roots in finite fields. *Math. Comp.*, 58:369–380, 1992.

[32] V. Shoup. *NTL: A Library for doing Number Theory*, 2014. Software, version 8.0.0. http://www.shoup.net/ntl.