Polynomial root finding over local rings and application to error correcting codes

Jérémy Berthomieu, Grégoire Lecerf, Guillaume Quintin

Applicable Algebra in Engineering, Communication and Computing manuscript No.
(will be inserted by the editor)

Polynomial root finding over local rings and application to error correcting codes

Jérémy Berthomieu · Grégoire Lecerf · Guillaume Quintin

Received: date / Accepted: date

Abstract This article is devoted to algorithms for computing all the roots of a univariate polynomial with coefficients in a complete commutative Noetherian unramified regular local domain, which are given to a fixed common finite precision. We study the cost of our algorithms, discuss their practical performances, and apply our results to the Guruswami and Sudan list decoding algorithm over Galois rings.

Keywords polynomial root finding · Galois ring · list decoding · Guruswami-Sudan algorithm

Mathematics Subject Classification (2000) 12Y05 · 94B05

1 Introduction

Throughout this paper, R denotes a *complete commutative Noetherian unramified regular local domain* of finite dimension r, with maximal ideal $\mathfrak{m}$. Let p denote the characteristic of the residue field $\kappa := R/\mathfrak{m}$ of R, and let $R_i := \mathfrak{m}^i/\mathfrak{m}^{i+1}$, for all $i \geqslant 0$.

Jérémy Berthomieu
UPMC, Univ. Paris 6, LIP6
INRIA Paris – Rocquencourt, POLSYS
CNRS UMR 7606, LIP6
Boîte courrier 169, 4 place Jussieu, 75252 Paris Cedex 5, France.
E-mail: jeremy.berthomieu@lip6.fr

Grégoire Lecerf
Laboratoire d'Informatique
CNRS UMR 7161, LIX
Campus de l'École polytechnique, 91128 Palaiseau Cedex, France.
E-mail: gregoire.lecerf@math.cnrs.fr

Guillaume Quintin
Laboratoire d'Informatique
CNRS UMR 7161, LIX
Campus de l'École polytechnique, 91128 Palaiseau Cedex, France.
E-mail: quintin@lix.polytechnique.fr

The fact that R is *unramified* means that either $p = 0$ holds, or that p does not belong to $\mathfrak{m}^2$. By [15, Theorem 15] the following alternative holds:

- If R and κ have the same characteristic whatsoever, then R is isomorphic to the *power series ring* $\kappa[[t_1,\ldots,t_r]]$. In this case, we identify R_i to the subgroup of R of the homogeneous polynomials in $t_1,\ldots,t_r$ over κ of degree i, so that $(R_i)_{i\in\mathbb{N}}$ defines a graduation on R.
- Otherwise, if R and κ have different characteristics, then R is isomorphic to the power series ring $D[[t_1,\ldots,t_{r-1}]]$, where D is a *complete discrete valuation ring* with maximal ideal generated by p. Each element of R can be uniquely written as $\sum_{e\in\mathbb{N}^r} c_e t_1^{e_1}\cdots t_{r-1}^{e_{r-1}} p^{e_r}$, with the c_e in κ. We can still identify R_i to the subset of R of the homogeneous polynomial expressions in $t_1,\ldots,t_{r-1}$ and p of degree i and with coefficients in κ, but $(R_i)_{i\in\mathbb{N}}$ does not define a graduation on R (for example with R being the ring of the p-adic integers $\mathbb{Z}_p$). In this case, we set $t_r := p$.

In both cases, the function $v : R \to \mathbb{N}\cup\{+\infty\}$, which sends 0 to $+\infty$, and any $a \neq 0$ to the largest integer i such that $a \in \mathfrak{m}^i$, is a *valuation*. Any element a of R can be uniquely represented by the converging sum $\sum_{i\geqslant 0}[a]_i$, where $[a]_i \in R_i$ is the *homogeneous component of valuation i* of a. The elements of R_i are called the *homogeneous elements of valuation i* of R.

In this paper we are interested in computing all the roots of a polynomial $F \in R[x]$ given to precision n, which means modulo $\mathfrak{m}^n$. The usual cases are for when $R = \mathbb{Z}_p$ or $R = \mathbb{K}[[t]]$, for any field $\mathbb{K}$. We will adapt classical techniques, analyze their cost, and report on practical performances of our C++ implementation using the MATHEMAGIX libraries [29].

1.1 Application to list decoding

Univariate polynomial root-finding is a central problem in computer algebra, and a major application resides in decoding certain error-correcting codes as recalled in these paragraphs. Let $a_1,\ldots,a_\lambda$ be λ distinct fixed points in the finite field with q elements, written $\mathbb{F}_q$. Let us recall that a *Reed-Solomon code* of length λ and dimension ρ over $\mathbb{F}_q$ is the set

$$\mathrm{RS}(\lambda,\rho) = \{(f(a_1),\ldots,f(a_\lambda)) : f \in \mathbb{F}_q[x]_{<\rho}\},$$

where $\mathbb{F}_q[x]_{<\rho}$ represents the set of polynomials over $\mathbb{F}_q$ of degree at most $\rho-1$ (we refer the reader for instance to [37, Chapter 6] for generalities on such codes).

This set $\mathrm{RS}(\lambda,\rho)$ is a vector subspace of $\mathbb{F}_q^\lambda$ of dimension ρ, and there is a one-to-one correspondance between polynomials of $\mathbb{F}_q[x]_{<\rho}$ and elements of $\mathrm{RS}(\lambda,\rho)$. To encode a message, the sender constructs the *unique* polynomial f of $\mathbb{F}_q[x]_{<\rho}$ corresponding to the message, and then transmits the vector $y = (f(a_1),\ldots,f(a_\lambda)) \in \mathbb{F}_q^\lambda$. The received vector may be different from y. If only a few errors occurred during the transmission of y, obtaining the original message can be done using the usual unambiguous decoding algorithms such as Berlekamp-Welch [8], Berlekamp-Massey [7], the extended Euclidean algorithms [44] and Gao's algorithm [19]. But, when more errors occur, a different decoding approach, called *list-decoding*, must be used. A

list-decoding algorithm outputs a set Y of possible transmitted messages. A postprocess is then responsible for deciding which element of Y is the actual message. Our present motivation lies in the list-decoding algorithms.

In [26], *Guruswami and Sudan* designed a polynomial-time list-decoding algorithm. Their method divides into two steps. First it computes a polynomial Q in $\mathbb{F}_q[x][y]$ such that the possible transmitted messages are roots of Q in $\mathbb{F}_q[x]$. In the second step one needs to determine all such roots of Q. Several techniques have been investigated to solve both steps of the problem: see for example [1,5,32,33] for the first step and [20, pages 214–228], and [20,41] for the second step.

The Guruswami and Sudan algorithm has been adapted to other families of codes such as algebraic-geometric codes over fields [26], and alternant codes over fields [4]. Extensions over certain types of finite rings have further been studied for Reed-Solomon and alternant codes in [2,3], and for algebraic-geometric codes in [6,45]. In all these cases, the two main steps of the Guruswami and Sudan algorithm are roughly preserved, but to the best of our knowledge, the second step has never been studied into deep details from the complexity point of view. In this paper, we investigate root-finding for polynomials over *Galois rings*, which are often used within these error correcting codes, and that are defined as follows:

Definition 1 Let $\varphi \in \mathbb{Z}/p^n\mathbb{Z}[x]$ be a monic polynomial of degree k that is irreducible modulo p. The ring $(\mathbb{Z}/p^n\mathbb{Z}[x])/(\varphi(x))$ is called the *Galois ring*, written $\mathrm{GR}(p^n,k)$, of *order* nk and *characteristic* p^n.

It is classical that there exists only one Galois ring of order nk and of characteristic p^n up to an isomorphism (see for example [39, p. 207]). On the other hand, notice that such a Galois ring can also be defined as $\mathrm{GR}(p^n,k) = R/(p^n)$, where R is an unramified algebraic extension $\mathbb{Z}_p$ of degree k. Over such a Galois ring $\mathrm{GR}(p^n,k)$ standard techniques cannot be applied to find all the roots of a given polynomial in $\mathrm{GR}(p^n,k)[t][x]$. For instance with $n=2$ and $F(x) = (x-p)(x-pt)$, one cannot find a value a for t that makes the specialization of F with a unit discriminant in the Galois ring, so that fast classical Newton-Hensel lifting cannot be appealed.

1.2 Complexity model

In order to analyze the performances of our algorithms, we denote by $\mathsf{M}(n)$ a cost function for multiplying two univariate polynomials of degree n over an arbitrary commutative ring A with unity, in terms of the number of arithmetic operations in A. Similarly, we denote by $\mathsf{I}(n)$ the time needed to multiply two integers of bit-size at most n in *binary representation*. It is classical [12,18,42] that we can take $\mathsf{M}(n) \in O(n\log n\log\log n)$ and $\mathsf{I}(n) \in O(n\log n 2^{\log^* n})$, where $\log^*$ represents the iterated logarithm of n. Throughout this paper, we assume that $\mathsf{M}(n)/n$ is increasing and that $\mathsf{M}(mn) \leqslant m^2\mathsf{M}(n)$ holds for all positive integers m and n. The same assumptions are also made for I.

When needed, we shall assume that root-finding is computable over the residue field κ. Let us recall here that there exist *effective fields* (that are defined as fields with an effective equality test) for which root-finding is not decidable [17, Section 7] (see

also another example in [21, Remark 5.10]). Hopefully in most practical cases, roots can be computed efficiently, as we shall recall it later over finite fields.

Finally, let us recall that the *expected cost* spent by a randomized algorithm is defined as the average cost for a given input over all the possible executions. The "soft-Oh" notation $f(n) \in \tilde{O}(g(n))$ means that $f(n) \in g(n)\log^{O(1)}(3+g(n))$ (we refer the reader to [22, Chapter 25, Section 7] for details).

1.3 Our contributions

Let $K := \mathrm{Quot}(R)$ represent the *total field of fractions* of R. Since R is supposed to be complete, so is K, and we still write v for the extension of the valuation from R to K. The subset of the elements of K of valuation at least i is written O_i. If a is an element of K, and if i is an integer, then we write $a+O_i$ for the set of elements in K whose expansion coincides to those of a to precision i. We say that such a *class* $a+O_i$ is a root of F to precision n if all of its elements annihilate F to precision n. Notice that, for all integers i and j, we have $O_i + O_j = O_{\min(i,j)}$. Thus for any two elements a and b in K we can write $(a+O_i)+(b+O_j) := (a+b)+O_{\min(i,j)}$. By convention, every element a of K can be seen as the class $a+O_\infty$, so that it makes sense to define the sum of an element of K to a class as follows: $a+(b+O_i) := (a+b)+O_i$.

The set of the roots of $F(x) = x^n$ in $\mathbb{Q}_p$ of nonnegative valuation and to precision n is made of all the elements of positive valuation, which amounts to p^{n-1} roots. This simple example shows that the number of roots can be exponential in terms of the size of F. However it can be represented by the single class O_1. In Section 2 we show that the roots of nonnegative valuation and to precision n of a polynomial $F \in O_0[x]$ of degree d can be represented by at most d such classes, in the sense that the set of roots equals the union of the elements in these classes. As another example, with $R = \mathbb{Z}_p$, the roots of nonnegative valuation and to precision 4 of $F(x) = x^2(x-1)$ are either 1 or an element of valuation at least 2 in $\mathbb{Q}_p$, that is in O_2.

Section 2 contains a "naive" algorithm for computing all the roots z of valuation at least a given nonnegative integer w and to a given precision n of a polynomial $F \in O_0[x]$. This algorithm first determines all the possible values for $[z]_w$. Then, from such a value $[z]_w$, it computes the shifted polynomial $F([z]_w + x)$ and it calls itself recursively to obtain the roots of valuation at least $w+1$. We analyze the complexity of this technique: in particular we show that all subparts but the shifts behave essentially in an optimal way. We also provide the reader with detailed complexity results when R is a univariate power series ring or the p-adic integers ring.

In Section 3 we modify the naive algorithm so that it splits the input polynomial between the recursive calls by Hensel lifting. In fact we extend the classical Hensel lifting to the quasi-homogeneous setting, and estimate how it decreases the cost of the previous "naive" algorithm. We detail complexity bounds when R is a univariate power series ring or the p-adic integers, but also exhibit a probabilistic fast version in higher dimension that avoids expression swell.

Section 4 is devoted to applying our root finders in the context of list decoding over Galois rings. We have implemented the algorithms of the present paper when R has Krull dimension 1 in the open source library QUINTIX of the MATHEMAGIX

computer algebra system [29]. We report on timings and discuss their relative performances.

1.4 Related works

Besides the aforementioned works in error correcting codes let us briefly discuss the known materials for computing roots of univariate polynomials over some particular instances of R as defined from the beginning of the present paper. In both theory and practice, it is classical to compute the factorization, or all the roots in an algebraic closure of a given polynomial $F \in R[x]$ for particular cases. The easiest case is for when the degree of F does not drop modulo $\mathfrak{m}$ and when F is separable modulo $\mathfrak{m}$: Hensel lifting leads efficiently to the unique factorization of F to any requested precision (we refer the reader for instance to [22, Chapter 15]). In general, even if F is separable, its residue polynomial modulo $\mathfrak{m}$ may have multiple factors, and one has to make use of the *Newton polygon* technique recursively, assuming that the characteristic is sufficiently large. Over the power series, namely when $R = \mathbb{K}[[t]]$, several authors have contributed to this subject including, for instance: [13,14,16,27,28,38, 46,47,48]. Over the p-adic integers the situation becomes more problematic but some of the latter techniques can be extended as in [27]. The case for when R is a power series ring in at least two variables has also been studied in [30,34]. In addition, for univariate power series in small characteristic, we refer the reader to [25,31]. In fact, all these techniques do not solve directly our problem over a general coefficient ring R as considered here, and not even in elementary situations as demonstrated by the following examples:

Example 1 Let $R = \mathbb{Z}_p$, and let $F(x) = (x-p)(x+p)$. In R the polynomial F admits two simple roots p and $-p$, but the set of roots modulo p^2 is the ideal (p). This shows that computing the roots of F in $\mathbb{Z}_p$ does not lead to the ones modulo p^2 directly. In addition the fact that 0 is a root modulo p^2 is contributed by the positive valuation of the values of both factors of F. This suggests that, in general, a kind of exhaustive search might be necessary to recover the modular roots from an irreducible factorization of F in R.

Example 2 Let $R = \mathbb{Z}_p$. The polynomial $F(x) = x^2$ admits 0 as a single double root, but the roots modulo p^4 form the ideal (p^2). Again this shows that there is no obvious relationship between the roots in $\mathbb{Z}_p$ and the ones in $\mathbb{Z}_p/(p^4)$.

These examples illustrate the difficulties for deducing the roots in the ring $R/\mathfrak{m}^n$ from the ones in R to a sufficiently large precision, or from an irreducible factorization over R. The ingredients of the present paper are not substantially new: our main contribution relies in the design of general and well-suited algorithms to the specific root-finding problem.

2 Algorithm with linear convergence

Recall that $K := \mathrm{Quot}(R)$ represents the *total field of fractions* of R. Since R is supposed to be complete, so is K, and we still write v for the extension of the valuation

from R to K. Any element a of K can be uniquely written as the sum $\sum_{i \geqslant v(a)} [a]_i$, where $[a]_i$ is 0 or has valuation i and is the quotient of two homogeneous elements in R. For any $i \in \mathbb{Z}$, we write K_i for the set of the elements $a \in K$ such that either a is 0 or a has a single component of valuation i, which means that $a = [a]_i$. The subset of the elements of K of valuation at least i is written O_i.

Definition 2 Let $F(x) = \sum_{l=0}^{d} F_l x^l \in K[x]$ be a polynomial of degree d. For any $w \in \mathbb{Z}$, *the w-homogeneous component of w-valuation i of F* is the polynomial $[F]_{i,w}$ such that

$$[F]_{i,w} := \sum_{l=0}^{d} [F_l]_{i-wl} x^l.$$

Polynomial F is said to be *w-homogeneous of w-valuation i*, whenever $F = [F]_{i,w}$. In addition, the expression $[F]_{j\ldots j+k,w}$ is used to represent the sum $\sum_{l=0}^{k-1} [F]_{j+l,w}$.

The quantity $v_w(F)$, called the *w-valuation* of F, stands for the first index $i \in \mathbb{Z}$ such that $[F]_{i,w}$ is nonzero, with the convention that $v_w(0) := +\infty$.

Remark that if $a \in K$ has valuation at least w then $[F]_{i,w}(a)$ has valuation at least i.

Example 3 For $R = \mathbb{Q}[[t]]$, and for $F = x^3 - (1+t)x^2 + t^3$, we have that $v_{-1}(F) = -3$, $[F]_{-3,-1} = x^3$, and that $v_0(F) = 0$, $[F]_{0,0} = x^3 - x^2$.

2.1 Local multiplicities

In this subsection we define the multiplicity of an homogeneous root of a w-homogeneous polynomial.

Lemma 1 (Quasi-homogeneous Euclidean division). *Let $H \in K[x]$ be a non-constant w-homogeneous polynomial of w-valuation i, and let $z \in K_w$. Then there exists a unique w-homogeneous polynomial $Q \in K[x]$ of w-valuation $i - w$, and a unique element $a \in K_i$, such that:*

$$H(x) = [(x-z)Q(x) + a]_{i,w}.$$

Proof When performing the classical long division of $H(x)$ by $x - z$ the w-homogeneity is preserved in w-valuation i when discarding the carries. □

From the latter lemma, if H is a w-homogeneous polynomial of w-valuation i, then it makes sense to define the *multiplicity* m of any $z \in K_w$ of H, written $\mathrm{mult}(z, H)$, as the largest integer m such that H rewrites into $[(x-z)^m Q(x)]_{i,w}$, where $Q \in K[x]$ is a w-homogeneous polynomial of w-valuation $i - mw$.

Lemma 2 *If $H \in K[x]$ is a nonzero w-homogeneous polynomial of w-valuation i, then the following inequality holds:*

$$\sum_{z \in K_w, H(z) \in O_{i+1}} \mathrm{mult}(z, H) \leqslant \deg H.$$

Proof Let $z \in K_w$ be of multiplicity m in H, and let $Q \in K[x]$ be as above. If $y \in K_w$ is a distinct root of H to precision $i+1$, then we have that

$$mv(y-z) + v(Q(y)) \geqslant i+1.$$

It follows that $v(Q(y)) \geqslant i-mw+1$, hence that y is a root of Q to precision $i-mw+1$. By a straightforward induction, we deduce that if $z_1, \ldots, z_s$ are the roots of H in K_w to precision $i+1$ then H factors into $[(x-z_1)^{m_1} \cdots (x-z_s)^{m_s} G(x)]_{i,w}$, where G is a w-homogeneous polynomial of w-valuation $i-w(m_1+\cdots+m_s)$, whence the claimed inequality. □

2.2 Representation of the set of roots

In this subsection we deal with the representation of sets of truncated roots.

Lemma 3 *Let F be a nonzero polynomial in $K[x]$ of $(w-1)$-valuation i, let $m := \mathrm{mult}(0, [F]_{i,w-1})$, and let $j := \nu_w(F)$. Then we have $i \leqslant j \leqslant i+m$, and $\deg[F]_{j,w} \leqslant j-i \leqslant m$. In addition, $\deg[F]_{j,w} = m$ holds if, and only if, $j = i+m$. In this case the leading coefficients of $[F]_{i,w-1}$ and of $[F]_{j,w}$ coincide.*

Proof From the assumptions we can express F as $F(x) = x^m Q(x) + H(x)$, where $Q \in K[x]$ is a $(w-1)$-homogeneous polynomial of $(w-1)$-valuation $i-m(w-1)$, such that $Q(0) \neq 0$, and where $H \in K[x]$ is a polynomial of $(w-1)$-valuation at least $i+1$. We see that F has a term ax^m with $\nu(a) = i-m(w-1)$ and $[a]_{i-m(w-1)} = Q(0)$. It follows that the w-valuation j of F is at most $i-m(w-1)+mw = i+m$. On the other hand, since a term of degree $k \geqslant j-i+1$ in F has $(w-1)$-valuation at least i, it contributes to w-valuation at least $i+k \geqslant j+1$. Therefore, no monomial of degree at least $j-i+1$ of F contributes to $[F]_{j,w}$.

If $\deg[F]_{j,w} = m$, then it is clear that $j-i = m$. Conversely, if $j-i = m$ then $[F]_{j,w}$ has the term $[a]_{i-m(w-1)} x^m$, hence has degree m. □

Although the next lemma is elementary, it constitutes the cornerstone of the solver presented in the next subsection.

Lemma 4 *Let F be a nonzero polynomial in $K[x]$ of w-valuation j. Then $a \in K$ is a root of valuation at least w of F to precision n if, and only if, $[F]_{j,w}([a]_w)$ vanishes to precision $j+1$ and $a-[a]_w$ is a root of valuation at least $w+1$ of $F([a]_w + x)$ to precision n.*

Proposition 1 *If F is a polynomial in $O_0[x]$ of w-valuation $j \leqslant n-1$, then its set of roots in K of valuation at least $w \geqslant 0$ and to precision n can be written as the disjoint union of at most $\deg[F]_{j,w}$ classes of the form $a + O_i$.*

Proof The proof is done by descending induction on w from n. If $w \geqslant n$ then the statement clearly holds since $\deg[F]_{j,w}$ becomes necessarily 0. Let us now assume by induction that the proposition holds for valuation $w+1 \leqslant n$. Let $z \in K_w$ be such that $[F]_{j,w}(z) \in O_{j+1}$, and let $m_z := \mathrm{mult}(z, [F]_{j,w})$. By Lemma 4 the number of classes of roots of F with z as initial term is the number of classes of roots of $F(z+x)$ with

valuation at least $w+1$ to precision n. If $j_z := v_{w+1}(F(z+x)) \geqslant n$, then there is only one such class. Otherwise the induction hypothesis ensures us that the number of classes is as most $\deg[F(z+x)]_{j_z,w+1}$, which is bounded by m_z by Lemma 3. The conclusion thus follows from Lemma 2. □

2.3 Naive local solver

We are to describe an algorithm derived from the proof of Proposition 1. For computational purposes, we need to assume that there exists an algorithm which computes the set of roots in K_w of any w-homogeneous polynomial $H(x)$, together with their respective multiplicities.

Algorithm 1

Input A polynomial $F \in O_0[x]$, $w \in \mathbb{N}$, $i \in \mathbb{N}$, $m \in \mathbb{N}$, $c \in K_{i-(w-1)m}$, and $n \in \mathbb{N}$, such that $i = v_{w-1}(F) \leqslant n-1$, $m = \text{mult}(0, [F]_{i,w-1}) \geqslant 1$, and c is the coefficient of degree m in $[F]_{i,w-1}$.

Output A set of at most m disjoint classes representing the roots of F in K with valuation at least w and to precision n.

1. Search for the first nonzero w-homogeneous component $\tilde{H}$ of F taken modulo x^m, of w-valuation k, with $i+1 \leqslant k \leqslant \min(i+m-1, n-1)$.
   a. If such a component $\tilde{H}$ does exist then
      set $j := k$ and $H := \tilde{H} = [F]_{j,w}$
      else
      if $i+m \leqslant n-1$ then set $j := i+m$, $\tilde{H} := [F]_{j,w} \bmod x^m$, and $H := \tilde{H} + cx^m = [F]_{j,w}$, otherwise return $\{O_w\}$.
   b. If H has degree 0 then return $\{\}$.
2. Compute all the roots $z_1, \ldots, z_s$ in K_w of H to precision $j+1$, together with their respective multiplicities $m_1, \ldots, m_s$.
3. For each e in $1, \ldots, s$ do
   a. Compute $F_e := F(z_e + x)$.
   b. If $m_e = m$ then let $c_e := c$. Otherwise set c_e to the coefficient of degree m_e in $[F_e]_{j,w}$.
   c. Call Algorithm 1 recursively with entries F_e, $w+1$, j, m_e, c_e, and n, in order to obtain the set $Z_{w+1,z}$ representing the roots of F_e of valuation at least $w+1$ to precision n.
4. Return $\{z + z' | z \in Z_w, z' \in Z_{w+1,z}\}$.

Proposition 2 *Algorithm 1 works correctly as specified. In addition, the polynomial H in step 2 of Algorithm 1 equals $[F]_{j,w}$.*

Proof The algorithm exits at step 1.a with $\{O_w\}$ whenever $v_w(F) \geqslant n$, which is correct. It exits at step 1.b with the empty set whenever H is a constant, which is also correct since $H = [F]_{j,w}$ by Lemma 3.

Then the proof is done by descending induction on w. If $w \geqslant n$ then the algorithm necessarily exits at step 1. Let us now assume that the proposition holds for $w+1 \leqslant n$.

By Lemma 3 again we have that $H = [F]_{j,w}$. In step 3.b, if $m_e = m$, then Lemma 3 guarantees that c is actually the coefficient of degree m in $[F]_{j,w}$, and thus of $[F_e]_{j,w}$. Therefore the correctness follows from Lemma 4. □

Example 4 Take $R = \mathbb{Q}[[t]]$. The trace of Algorithm 1 with input $F(x) = x^3 - (1+t)x^2 + t^3$, $w = 0$, $i = -3$, $m = 3$, $c = 1$, and $n = 4$ is the following:

1. No $\tilde{H}$ is found with w-valuation in $\{-2,-1\}$. Since $i+m = 0 \leqslant 3 = n-1$, we have $j = i+m = 0$ and $H(x) = x^3 - x^2$.
2. $z_1 = 0$, $m_1 = 2$, $z_2 = 1$, $m_2 = 1$.
3. Algorithm 1 is called recursively with input $F(0+x) = x^3 - (1+t)x^2 + t^3$, $w = 1$, $i = 0$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
   1. $j = 2$ and $H(x) = -x^2$.
   2. $z_1 = 0$, $m_1 = 2$.
   3. Algorithm 1 is called recursively with input $F(0+x) = x^3 - (1+t)x^2 + t^3$, $w = 2$, $i = 2$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
      1. $j = 3$, $H(x) = t^3$, and the algorithm returns $\{\}$.
   4. The algorithm returns $\{\}$.

   Algorithm 1 is then called recursively with input $F(1+x) = x^3 + (2-t)x^2 + (1-2t)x - t + t^3$, $w = 1$, $i = 0$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
   1. $j = 1$ and $H(x) = x - t$.
   2. $z_1 = t$, $m_1 = 1$.
   3. Algorithm 1 is called recursively with input $F(1+t+x) = x^3 + 2(2+t)x^2 + (1+2t+t^2)x + t^3$, $w = 2$, $i = 1$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
      1. $j = 2$ and $H(x) = x$.
      2. $z_1 = 0$, $m_1 = 1$.
      3. Algorithm 1 is called recursively with input $F(1+t+x) = x^3 + 2(1+t)x^2 + (1+2t+t^2)x + t^3$, $w = 3$, $i = 2$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
         1. $j = 3$ and $H(x) = x + t^3$.
         2. $z_1 = -t^3$, $m_1 = 1$.
         3. Algorithm 1 is called recursively with input $F(1+t-t^3+x) = x^3 + (2+2t-3t^3)x^2 + (1+2t+t^2-4t^3-4t^4+3t^6)x - 2t^4 - t^5 + 2t^6 + 2t^7 - t^9$, $w = 4$, $i = 3$, $c = 1$, and $n = 4$, and runs as follows:
            1. The algorithm returns $\{O_4\}$.
         4. The algorithm returns $\{-t^3 + O_4\}$.
      4. The algorithm returns $\{t - t^3 + O_4\}$.
4. The algorithm finally returns $\{1 + t - t^3 + O_4\}$.

2.4 Cumulative cost of steps 1

In step 1 of Algorithm 1, we are interested in counting the cumulative number of extractions of quasi-homogeneous components, and zero tests performed in each graduated component of K. For this purpose we introduce the following subset $T_{i,w,m,n}$ of $\mathbb{N}^2$:

$$T_{i,w,m,n} := \left\{ (k,l) \in \mathbb{N}^2 \middle| k \leqslant m-1 \text{ and } l \leqslant n-1 \text{ and } wk + l \geqslant i+1 \right\}.$$

For any subset S of $\mathbb{N}^2$, we write $|S|$ for its cardinality, and $[S]_v$ for $S \cap (\mathbb{N} \times \{v\})$. Roughly speaking, the following lemma ensures us that the cumulative cost of steps 1 of Algorithm 1 is essentially optimal, whenever an element $a \in O_0$ to precision n is represented as a vector in $K_0 \times \cdots \times K_{n-1}$:

Lemma 5 *For all $v \in \{0,\ldots,n-1\}$, the cumulative number of extractions of homogeneous components of valuation v and the cumulative number of zero tests in each K_v in all steps 1 of Algorithm 1 is at most $|[T_{i,w-1,m,n}]_v| \leqslant m$.*

Proof The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then step 1.a extracts all the components of valuation l of the constant coefficient of F, for $l \geqslant i+1$. The statement therefore holds in this case since $m \geqslant 1$.

Assume that the lemma holds for $w+1 \leqslant n$. We introduce the auxiliary subset of $\mathbb{N}^2$:

$$S_0 := \{(k,l) \in T_{i,w-1,m,n} | wk+l \leqslant j\}.$$

In step 1 of Algorithm 1 only the components of valuation l of the coefficients of x^k for (k,l) in S_0 need to be examined.

Let $M_e := m_1 + \cdots + m_{e-1}$, with the usual convention that $M_1 := 0$. Then, each recursive call for $F(z_e + x)$ in step 3 amounts to at most $|[S_e]_v|$ component extractions and zero tests in K_v, where

$$S_e := (M_e, 0) + T_{j,w,m_e,n}, \text{ for all } e \in \{1,\ldots,s\}.$$

Notice that $S_e \subseteq T_{i,w-1,m,n}$ holds for all $e \geqslant 0$ by using Lemma 3. On the other hand the S_e are pairwise disjoint. Therefore we obtain that $\sum_{e=0}^{s} |[S_e]_v| \leqslant |[T_{i,w-1,m,n}]_v|$, which concludes the proof. □

2.5 Cumulative cost of steps 2

The following proposition concerns the sum of the degrees of all the polynomials H occurring during the execution of Algorithm 1.

Lemma 6 *The sum of the degrees of all the polynomials H occurring during the execution of all steps 2 of Algorithm 1 does not exceed $m \max(0, n-w)$.*

Proof The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then the statement is true since Algorithm 1 exits at step 1. Let us now assume by induction that the lemma holds for $w+1 \leqslant n$. By Lemma 3, each recursive call in step 3 performs root-finding of polynomials whose degree sum does not exceed $m_e(n-(w+1))$. The conclusion thus follows thanks to Lemma 2 as follows:

$$\begin{aligned} m + \sum_{e=1}^{s} (n-(w+1)) m_e &= (n-w)m - (n-(w+1)) \left(m - \sum_{e=1}^{s} m_e \right) \\ &\leqslant (n-w)m. \end{aligned}$$

□

2.6 Cumulative cost of steps 3

Let A be any ring. The *shift* of a polynomial $F \in A[x]$ at a point $a \in A$ is the computation of $F(a+x)$. We write $\mathscr{S}_A(d)$ for a bound on the cost of the shift in degree d for $F \in A[x]$ in terms of the number of arithmetic operations in A. We assume that $\mathscr{S}_A(d)/d$ is increasing and that $\mathscr{S}(md) \leqslant m^2\mathscr{S}(d)$ holds for all positive integers m and d. For the sake of completeness, we briefly recall a classical complexity bound:

Lemma 7 *Let A be a commutative ring with unity, let $F \in A[x]$ be a polynomial of degree d, and let $a \in A$. Then the computation of the shifted polynomial $F(a+x)$ can be done with $O(\mathsf{M}(d)\log d)$ operations in A.*

Proof We apply the classical divide-and-conquer paradigm. Without loss of generality we can assume that d is a power of 2. We rewrite $F(x)$ into $F_0(x) + x^{d/2}F_1(x)$, with $F_0, F_1 \in A[x]$ of degree at most $d/2$, so that we have $F(a+x) = F_0(a+x) + (a+x)^{d/2}F_1(a+x)$. First we compute all the successive powers $(a+x)^{2^1}, (a+x)^{2^2}, \ldots, (a+x)^{d/2}$, which amounts to $O(\mathsf{M}(d))$ operations in A. Then, the result classically follows from solving the recurrence $\mathscr{S}_A(d) \in 2\mathscr{S}_A(d/2) + O(\mathsf{M}(d/2))$, and the assumptions on M. □

Remark 1 Let us mention that the shifted polynomial can be computed faster in some situations. For instance, if $2, 3, \ldots, d$ are invertible in A, and if their respective inverses are given, then one has $\mathscr{S}_A(d) \in O(\mathsf{M}(d))$ by [10, Chapter 1, Section 2]. For situations in positive characteristic where the shift can be done within $O(\mathsf{M}(d))$, we refer the reader to [11, Proposition 5].

Lemma 8 *Algorithm 1 performs at most $m\max(0, n-w)$ shifts in $O_0[x]$ to precision n.*

Proof The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then no shift is performed, so the lemma is true. Let us assume that the lemma holds for $w+1 \leqslant n$. The combination of Lemmas 2 and 3 tells us that the cumulative number of the shifts spent by Algorithm 1 in all steps 3 is at most

$$s + \sum_{e=1}^{s} (n - (w+1))m_e \leqslant (n-w)m + s - \sum_{e=1}^{s} m_e \leqslant (n-w)m.$$

□

For steps 3.b we proceed as for steps 1. We introduce the following subset $T'_{i,w,m,n}$ of $\mathbb{N}^2$:

$$T'_{i,w,m,n} := \left\{(k,l) \in \mathbb{N}^2 | 1 \leqslant k \leqslant m, l \leqslant n-1 \text{ and } wk + l \geqslant i+1\right\}.$$

The following lemma ensures us that the cumulative cost of steps 3.b of Algorithm 1 is essentially optimal, whenever an element $a \in O_0$ to precision n is represented as a vector in $K_0 \times \cdots \times K_{n-1}$:

Lemma 9 *For all $v \in \{0,\dots,n-1\}$, the cumulative number of extractions of homogeneous components of valuation v and the cumulative number of zero tests in each K_v in all steps 3.b of Algorithm 1 are at most $|[T'_{i,w-1,m,n}]_v| \leqslant m$.*

Proof The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then the lemma clearly holds since the algorithm exits in step 1. Assume that the lemma holds for $w+1 \leqslant n$, and let $M_e := m_1 + \cdots + m_e$, for $e \in \{1,\dots,s\}$. If $e = 1$ and $m_1 = m$ then we set $S'_0 := \{\}$, otherwise we set $S'_0 := \{(M_e, j - wm_e) | e = 1,\dots,s\}$. In step 3.b, when $e \neq 1$ or $m_1 \neq m$ then we associate the component of valuation $j - wm_e$ of the coefficient of x^{m_e} to the point $(M_e, j - wm_e)$ in S'_0.

Then each recursive call for $F(z_e + x)$ in step 3.c amounts to $|[S'_e]_v|$ component extractions and zero tests in K_v, where $S'_e := (M_{e-1}, 0) + T'_{j,w,m_e,n}$, for all $e \in \{1,\dots,s\}$. Finally notice that $S'_e \subseteq T'_{i,w-1,m,n}$ holds for all $e \geqslant 0$ and that all the S'_e are pairwise disjoint. □

2.7 Cumulative cost of steps 4

Lemma 10 *The cumulative number of additions of an element of K_v to an element of $K_{v+1} \times \cdots \times K_{n-1}$ performed in all steps 4 of Algorithm 1 is 0 for $v \leqslant w-1$ and at most m for $v \geqslant w$.*

Proof We prove the lemma by descending induction on w from n down to 0. If $w \geqslant n$ then the lemma is true since step 4 is not reached. Let us now assume by induction that the lemma holds for $w+1 \leqslant n$. If $j \geqslant n$ or if H is a constant then step 4 is not executed. Otherwise by induction and Lemmas 2 and 3, all the recursive calls to Algorithm 1 in step 3 amount to at most m additions of an element of K_v to an element of $K_{v+1} \times \cdots \times K_{n-1}$ if $v \geqslant w+1$ and 0 otherwise. Then step 4 performs at most m additions of an element of K_w to an element of $K_{w+1} \times \cdots \times K_{n-1}$, which concludes the proof. □

2.8 Total cost of Algorithm 1

We assume that κ has either characteristic zero, or admits an algorithm that, for any $k \in \mathbb{N}$, detects if a given element is a p^kth power or not, and returns its p^kth root if it exists. We call this task an *iterated pth root extraction*. Let us recall that the *separable decomposition* of a primitive univariate non-constant polynomial G with coefficients in a unique factorization domain A is the decomposition of G into a product $G(x) = \prod_{i=1}^{s} G_i(x^{q_i})^{\mu_i}$, where

- the $G_i \in A[x]$ are primitive, separable, and have positive degrees,
- the $G_i(x^{q_i})$ are pairwise coprime,
- q_i is a power of p if $p > 0$, otherwise $q_i = 1$,
- μ_i is not divisible by p, and the (q_i, μ_i) are pairwise distinct.

The quantity $\sum_{i=1}^{s} \deg G_i$ is called the *separable degree* of G and is denoted by $\mathrm{sdeg}\, G$. Let us recall that the separable decomposition always exits and is unique up to permutation of the factors and units in A (see for instance [36, Proposition 4]). It coincides with the squarefree decomposition if A has characteristic 0.

From now on, for algorithmic purposes, any element a of R known to precision n is supposed to be stored in dense representation, as the vector $([a]_0, [a]_1, \ldots, [a]_{n-1})$. Any nonzero homogenous element c of valuation $v(c)$ is stored as a vector $(c_e)_{e\in\mathbb{N}^r}$ such that

$$c = \sum_{e\in\mathbb{N}^r, e_1+\cdots+e_r=v(c)} c_e t_1^{e_1}\cdots t_r^{e_r},$$

with all the c_e in κ. Recall that when R and κ have different characteristic then t_r represents p. For such an element c, we write $c^\flat$ for the expression

$$c^\flat := \sum_{e\in\mathbb{N}^r, e_1+\cdots+e_r=v(c)} c_e t_1^{e_1}\cdots t_{r-1}^{e_{r-1}} \in \kappa[t_1,\ldots,t_{r-1}],$$

obtained by substituting 1 for t_r syntactically. If $H(x) = \sum_{l=0}^{d} H_l x^l$ is a w-homogeneous polynomial then we further set $H^\flat(x) := \sum_{l=0}^{d} H_l^\flat x^l$.

Theorem 1 *For any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing primitive parts and separable decompositions of polynomials in $\kappa[t_1,\ldots,t_{r-1}][x]$ of degrees at most d in x and total degrees at most $n-1$ in $t_1,\ldots,t_{r-1}$, and whose degree sum is at most nd,*
- *computing roots in $\kappa[t_1,\ldots,t_{r-1}]$ of at most nd primitive polynomials of degrees 1 and total degrees at most $n-1$ in $t_1,\ldots,t_{r-1}$,*
- *computing roots in $\kappa[t_1,\ldots,t_{r-1}]$ of separable polynomials in $\kappa[t_1,\ldots,t_{r-1}][x]$ of degrees at least 2 and at most d, of total degrees at most $n-1$ in $t_1,\ldots,t_{r-1}$, and whose degree sum is at most $2(d-1)$,*
- *extracting iterated pth roots of at most $O(nd/p)$ elements in $\kappa[t_1,\ldots,t_{r-1}]$,*
- *$O(nd)$ shifts of polynomials in $R[x]$ of degree at most d and to precision n, and*
- *an additional number of $O(d)$ extractions of homogenous components of valuation v, and zero tests in each R_v, for each $v \in \{0,\ldots,n-1\}$.*

Proof Firstly we claim that running Algorithm 1 with input $F \in R[x]$ and finding the only roots in R_w instead of in K_w in step 2 actually leads to the set of roots in R of valuation at least w and to precision n. We leave the proof of this claim to the reader.

We enter this modified Algorithm 1 with input F, $w = 0$, $i = v_{-1}(F)$, $m = \mathrm{mult}(0, [F]_{i,-1})$, n, and the coefficient of degree m of $[F]_{i,-1}$. Determining the values of i and m takes no more than $O(d)$ extractions of homogenous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0,\ldots,n-1\}$. The cumulative costs of steps 1, 3.b and 4 of Algorithm 1 also drop into $O(d)$ such operations by Lemmas 5, 9, and 10 respectively.

Concerning step 2, we are looking for the roots $z \in R_w$ to precision $j+1$ of $H(x)$. If $H(z) \in O_{j+1}$ then $H^\flat(z^\flat) = 0$ holds in $\kappa[t_1,\ldots,t_{r-1}][x]$ and $z^\flat$ is a polynomial of

degree at most w. Conversely, if

$$y = \sum_{e \in \mathbb{N}^{r-1}} y_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} \in \kappa[t_1, \ldots, t_{r-1}]$$

has total degree at most w and is a root of $H^\flat(x)$, then we define

$$y^\natural := \sum_{e \in \mathbb{N}^{r-1}} y_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} t_r^{w-e_1-\cdots-e_{r-1}} \in R_w,$$

so that $H(y^\natural)$ belongs to O_{j+1}. Therefore, step 2 can be decomposed into the following tasks:

i. Compute the primitive part G of $H^\flat$ and the separable decomposition $G(x) = \prod_{i=1}^{s} G_i(x^{q_i})^{\mu_i}$ seen as in $\kappa[t_1, \ldots, t_{r-1}][x]$,
ii. Compute all the roots in $\kappa[t_1, \ldots, t_{r-1}]$ of all the latter $G_i(x)$,
iii. Extract the necessary q_ith roots of the roots of $G_i(x)$ in order to deduce the ones of $G_i(x^{q_i})$,
iv. Homogenize all the roots y found in iii with t_r, in valuation w, into $y^\natural$ as previously described.

The cumulative cost of tasks i and iii follows from Lemma 6. The cumulative cost of root-finding in ii of polynomials of degree at least 2 follows from Lemma 11 below. Finally the cumulative cost of the shifts in steps 3.a is deduced from Lemma 8. □

If $G_1, \ldots, G_r$ are polynomials, then we call the quantity $\sum_{e=1}^{r} (\operatorname{sdeg} G_e - 1)$ the *sum of the separable degrees minus* 1 of $G_1, \ldots, G_r$.

Lemma 11 *The sum of the separable degrees minus* 1 *of all the polynomials $G(x)$ of steps* i *in the proof of Theorem 1 is at most $m-1$.*

Proof The proof is done by descending induction on w. If $w \geqslant n$ then the lemma is true since $m \geqslant 1$ and the algorithm exits in step 1. Let us now assume that the lemma holds for $w+1 \leqslant n$. If the algorithm exits in step 1 then the lemma is correct. Otherwise, we let m_0 represent the separable degree of $G(x)$. Each recursive call to Algorithm 1 in step 3 performs root-finding of polynomials whose sum of the separable degrees minus 1 does not exceed $m_e - 1$. The total sum of the separable degrees minus 1 is at most

$$\begin{aligned} m_0 - 1 + \sum_{e=1}^{s} (m_e - 1) &\leqslant m_0 - 1 + \sum_{y \in \overline{\kappa(t_1,\ldots,t_{r-1})}, G(y)=0} (\operatorname{mult}(y, G) - 1) \\ &= m_0 - 1 + \deg G - m_0 \\ &\leqslant \deg G - 1. \end{aligned}$$

Finally Lemma 3 provides us with $\deg G - 1 \leqslant m - 1$. □

Corollary 1 *Let $\mathbb{K}$ be a field, and let R be the power series ring $\mathbb{K}[[t]]$. Then, for any polynomial F in $R[x]$ of degree at most d and given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in* $\mathbb{K}$ *of separable polynomials in* $\mathbb{K}[[x]]$ *of degrees at least* 2*, and whose degree sum is at most* $2(d-1)$*,*
- *extracting iterated pth roots of at most* $O(nd/p)$ *elements in* $\mathbb{K}$*, and*
- *an additional number of* $O(nd\mathsf{M}(n)\mathsf{M}(d)\log d)$ *arithmetic operations in* $\mathbb{K}$*.*

Proof This is a corollary of Theorem 1. In fact, by [36, Proposition 5], the cumulative cost of the separable factorizations amounts to $O(n\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. Finally, the cumulative cost of the shifts in steps 3.a is in $O(nd\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 7. □

Corollary 2 *Let* $\mathbb{K}$ *be a field of characteristic* 0 *and let R be the power series ring* $\mathbb{K}[[t]]$*. Then, for any polynomial F in* $R[x]$ *of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in* $\mathbb{K}$ *of separable polynomials in* $\mathbb{K}[[x]]$ *of degrees at least* 2*, and whose degree sum is at most* $2(d-1)$*, and*
- *an additional number of* $O(nd\mathsf{M}(n)\mathsf{M}(d))$ *arithmetic operations in* $\mathbb{K}$*.*

Proof This follows from the previous corollary, by means of Remark 1 that removes a factor of $\log d$ in the cost of the shifts. □

Corollary 3 *Let R be the power series ring* $\mathbb{F}_q[[t]]$ *over the finite field with* $q = p^k$ *elements. Then, for any polynomial F in* $R[x]$ *of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of*

$$O\left((nd\mathsf{M}(n)+\log q)\mathsf{M}(d)\log d+\frac{nd}{p}\log(q/p)\right)$$

operations in $\mathbb{F}_q$*.*

Proof By [22, Corollary 14.16] and Corollary 1, the cumulative cost for root-finding amounts to $O(\mathsf{M}(d)\log d\log(dq))$ operations in $\mathbb{F}_q$. □

Let us now focus on the case when R is an unramified algebraic extension of degree $k \geqslant 1$ of the ring $\mathbb{Z}_p$ of the p-adic integers. The ring $R/\mathfrak{m}^n$ is in fact the Galois ring, previously written $\mathrm{GR}(p^n,k)$, in Definition 1. We consider that we are given a monic irreducible polynomial φ in $\mathbb{Z}_p[z]$ of degree k. Let α denote the image of z in R viewed as $(\mathbb{Z}/p^n\mathbb{Z}[z])/(\varphi(z))$. Then, any $a \in R$ can be uniquely written as $\sum_{i=0}^{k-1} a_i\alpha^i$ with $a_i \in \mathbb{Z}/p^n\mathbb{Z}$. We further assume that each a_i is represented by its p-adic expansion $\sum_{j=0}^{n-1} a_{i,j}p^j$, which is stored as the vector $(a_{i,0},\ldots,a_{i,n-1})$ in $(\mathbb{Z}/p\mathbb{Z})^n$, and where each $a_{i,j}$ is in binary representation. It is classical that the bit-cost for multiplying two elements in $R/\mathfrak{m}^n$ falls in $\tilde{O}(nk\log p)$ [22, Chapter 9].

Corollary 4 *Let R be an unramified extension of* $\mathbb{Z}_p$ *of degree k. Then, for any given polynomial F in* $R[x]$ *of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of* $\tilde{O}((n^2d+\max(1,n/p)k\log p)dk\log p)$ *bit-operations.*

Proof This is again a corollary of Theorem 1. In fact, by [36, Proposition 5], the cumulative cost of the primitive parts and separable factorizations amounts to $\tilde{O}(nd)$ operations in $\mathbb{F}_q$, where $q := p^k$, which boils down to $\tilde{O}(ndk \log p)$ bit-operations. By [22, Corollary 14.16], the cumulative cost for root-finding amounts to $O(\mathsf{M}(d) \log d \log(dq))$ operations in $\mathbb{F}_q$, whence $\tilde{O}(d(k \log p)^2)$ bit-operations. The iterated root extractions take $O\left(\frac{nd}{p} \log \frac{q}{p}\right)$ operations in $\mathbb{F}_q$. Finally, the cumulative cost of the shifts in steps 3.a is in $\tilde{O}((nd)^2 k \log p)$ by Lemma 7. □

Remark 2 One could decide to store each a_i directly in binary representation modulo p^n: this does not change the latter asymptotic complexity estimate because the change of basis can be computed in softly linear time. In practice this does lightly increase the cost for extracting homogeneous components, but we have shown that these extractions are negligible compared to other operations. Let us mention here that recent practical algorithms on p-adic integers can be found in [9].

3 Faster algorithm with splitting

In most situations, the bottleneck of Algorithm 1 resides in the shifts applied on polynomials whose degrees never drop throughout the recursive calls. In this section, we enhance the solver of the previous section by adapting Hensel lifting in order to break the current polynomials into smaller pieces throughout each recursive call.

3.1 Quasi-homogenous Hensel lifting

For any real number $a \in \mathbb{R}$, we write $\lceil a \rceil$ for the smallest integer greater or equal to a. The quasi-homogeneous Hensel lifting algorithm for $F \in K[x]$ summarizes as follows:

Algorithm 2

Input Polynomials F, H_1, H_2, and U in $K[x]$, and integers $w \geqslant 0$, $j \geqslant 0$, and $l \geqslant 1$, such that:
- H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,
- H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,
- $[F]_{0 \ldots j+l,w} = [H_1 H_2]_{0 \ldots j+l,w}$,
- the resultant $\mathrm{Res}(H_1, H_2)$ has valuation $d_1 j_2 = d_1 d_2 w$,
- U has degree at most $d_1 - 1$, w-valuation $-j_2$, and $UH_2 = 1$ holds modulo H_1 and to w-precision $\lceil l/2 \rceil$.

Output H_1^*, H_2^*, and U^* in $K[x]$ such that:
- H_1^* is monic of degree d_1 and $[H_1^*]_{0 \ldots j_1+l,w} = [H_1]_{0 \ldots j_1+l,w}$,
- $[F]_{0 \ldots j+2l,w} = [H_1^* H_2^*]_{0 \ldots j+2l,w}$,
- $U^* H_2^* = 1$ holds modulo H_1^* and to w-precision l.

1. Compute $U^* := (2 - H_2 U)U$ modulo H_1 and to w-precision $-j_2 + l$.
2. Compute $\Delta_F := F - H_1 H_2$ to w-precision $j + 2l$.

3. Compute $\Delta_1 := U^*\Delta_F$ modulo H_1 and w-precision j_1+2l.
4. Set H_1^* to $H_1+\Delta_1$, and deduce $H_2^* := F/H_1^*$ to w-precision j_2+2l.

Algorithm 2 extends the classical Hensel lifting, which specifically concerns the case $w = j_1 = j_2 = 0$ (we refer the reader for instance to [22, Chapter 15, Section 4]).

Proposition 3 *Algorithm 2 works correctly as specified. Polynomial H_1^* (resp. H_2^*, U^*) is uniquely determined to w-precision j_1+2l (resp. j_2+2l, l) with the conditions required in the output.*

Proof It is straightforward to check that $U^*H_2 = 2UH_2-(UH_2)^2 = 1-(1-UH_2)^2 = 1$ holds modulo H_1 and to w-precision l. Let Δ_1 denote an unknown polynomial of w-valuation at least j_1+l, and let Δ_2 denote another unknown polynomial of w-valuation at least j_2+l. By expanding the right-hand side of the equation $F = (H_1+\Delta_1)(H_2+\Delta_2)$, we obtain that

$$F - H_1H_2 = H_2\Delta_1 + H_1\Delta_2 + \Delta_1\Delta_2.$$

Truncating the latter expression to w-precision $j+2l$ leads to

$$[F - H_1H_2]_{j+l\ldots j+2l,w} = [H_2\Delta_1 + H_1\Delta_2]_{j+l\ldots j+2l,w}.$$

By multiplying both hand sides of the latter equation by U^* modulo H_1, we deduce that:

$$[U^*(F - H_1H_2) \bmod H_1]_{j_1+l\ldots j_1+2l,w} = [\Delta_1 \bmod H_1]_{j_1+l\ldots j_1+2l,w}.$$

It follows that Δ_1 exists and is uniquely determined to w-precision j_1+2l. Therefore H_1^* exists and is uniquely determined as $H_1+\Delta_1$. Then H_2^* is necessarily determined as F/H_1^* truncated to w-precision j_2+2l. □

Example 5 Let $R = \mathbb{Z}_p[[t]]$, $F(x) = x^2-(p^2+t^2)x+p^2t^2+t^5$, $w = 2$, $j = 4$, $l = 1$, $H_1(x) = x-p^2$, and $H_2(x) = x-t^2$. We have $d_1 = d_2 = 1$, $j_1 = j_2 = 2$, and $j = 4$. The modular inverse U is $1/(p^2-t^2)$. Since one has the Bézout identity $\frac{1}{p^2-t^2}H_2(x) - \frac{1}{p^2-t^2}H_1(x) = 1$, then $U^* = 1$. We compute $\Delta_F = t^5$, then $\Delta_1 = t^5/(p^2-t^2)$, and obtain $H_1^*(x) = x-p^2+t^5/(p^2-t^2)$. Then, performing the Euclidean division on $F(x)$ and $H_1^*(x)$ at w-precision j_2+2l yields $H_2^*(x) = x-t^2-t^5/(p^2-t^2)$.

Before calling several times Algorithm 2 in order to reach any finite w-precision $j+l$ from w-precision j, one must compute the modular inverse of H_2 modulo H_1, and proceed as summarized in the next algorithm:

Algorithm 3

Input Polynomials F, H_1, and H_2 in $K[x]$, and integers $w \geqslant 0$, $j \geqslant 0$, and $n \geqslant 1$, such that:
- H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,
- H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,
- $[F]_{j,w} = [H_1H_2]_{j,w}$,
- the resultant $\mathrm{Res}(H_1,H_2)$ has valuation $d_1j_2 = d_1d_2w$.

Output H_1^*, H_2^* in $K[x]$ such that:

- H_1^* is monic of degree d_1 and $[H_1^*]_{j_1,w} = [H_1]_{j_1,w}$,
- $[F]_{0\ldots j+n,w} = [H_1^* H_2^*]_{0\ldots j+n,w}$.

1. Compute the inverse U of H_2 modulo H_1 in w-valuation $-j_2$.
2. Let $l := 1$, $U^* := U$, $H_1^* := H_1$, and $H_2^* := H_2$.
3. While $l < n$ do
   a. Call Algorithm 2 with F, H_1^*, H_2^* and U^*, w, j, and l, and reassign the output into H_1^*, H_2^* and U^* respectively.
   b. $l := \min(2l, n)$.
4. Return H_1^* and H_2^*.

Proposition 4 *Algorithm 3 works properly as specified.*

Proof Since $\mathrm{Res}(H_1, H_2)$ has valuation $d_1 j_2$, the valuation of the inverse of H_2 modulo H_1 as computed in step 1 is exactly $-j_2$. The rest of the proof follows from Proposition 3. □

Corollary 5 *Let F, H_1, and H_2 in $K[x]$ be such that the following conditions hold:*

- *H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,*
- *H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,*
- *$[F]_{j,w} = [H_1 H_2]_{j,w}$,*
- *the resultant $\mathrm{Res}(H_1, H_2)$ has valuation $d_1 j_2$.*

Then there exist unique polynomials H_1^* and H_2^* in $K[x]$ such that:

- *H_1^* is monic of degree d_1, has w-valuation j_1, and $[H_1^*]_{j_1,w} = [H_1]_{j_1,w}$,*
- *$F = H_1^* H_2^*$.*

In addition, if F belongs to $R[x]$ then $H_2^*(z)H_1^*$ also belongs to $R[x]$, for all $z \in R_w$.

Proof The existence of H_1^* and H_2^* immediately follows from Proposition 4 since K is complete. As for the last statement, let $z \in R_w$, and let m represent the multiplicity of z in H_1^* ($m = 0$ if z is not a root of H_1^*), let $\tilde{F}(x) := F(x)/(x-z)^m$ and let $\tilde{H}_1^*(x) := H_1^*(x)/(x-z)^m$. Since R is factorial by [15, Theorem 18], Gauss's lemma [35, Chapter IV, Theorem 2.1] ensures us that $\tilde{F}(z)\tilde{H}_1^*/\tilde{H}_1^*(z)$ belongs to $R[x]$. But the latter expression precisely rewrites into $H_2^*(z)\tilde{H}_1^*$, whence $H_2^*(z)H_1^* \in R[x]$. □

Algorithm 2 takes $O(\mathsf{M}(\deg F))$ operations in K. A general cost analysis in terms of operations in κ is difficult since it involves bounding sizes of numerators and denominators of the elements in K used during the intermediate computations. Concerning Algorithm 3, one must in addition describe how the modular inverse of H_2 modulo H_1 is actually obtained. For these reasons, from now on we restrict to considering that the elements of R are represented as in Section 2.8. We focus on the important case of dimension 1. Higher dimension is studied in Section 3.6.

Lemma 12 *Assume that R has dimension $r = 1$, and let F be a polynomial in $R[x]$ of degree at most d. Then Algorithm 3 can be run so that it performs $O(\mathsf{M}(d)\log d)$ operations in κ, and $O(\mathsf{M}(d))$ operations in $R/\mathfrak{m}^l$, for each value of l in the set $\{1, 2, 4, \ldots, 2^\lambda | 2^\lambda < n\} \cup \{n\}$.*

Proof The simplest way to implement Algorithm 3 in dimension 1 is to compute $\tilde{F}(x) := F(t_r^w x)/t_r^j$, $\tilde{H}_1(x) := H_1(t_r^w x)/t_r^{j_1}$, $\tilde{H}_2 := H_2(t_r^w x)/t_r^{j_2}$, and $\tilde{U} := U(t_r^w x)/t_r^{-j_2}$, and call Algorithm 3 with input $\tilde{F}$, $\tilde{H}_1$, $\tilde{H}_2$, $w = 0$, $j = 0$, and n. Step 1 can thus be performed by computing an extended g.c.d. between $\tilde{H}_1$ and $\tilde{H}_2$ modulo t_r, which takes $O(\mathsf{M}(d)\log d)$ operations in κ by [22, Corollary 11.8]. Then each call to Algorithm 2 can be performed with $O(\mathsf{M}(d))$ operations in R to precision l. Of course at the end we recover H_1^* as $\tilde{H}_1^*(x/t_r^w)t_r^{j_1}$ and H_2^* as $\tilde{H}_2^*(x/t_r^w)t_r^{j_2}$. □

3.2 Quasi-homogeneous multifactor Hensel lifting

In this subsection we appeal to the classical divide and conquer paradigm in order to lift any factorization of F into s factors in an efficient way.

Algorithm 4

Input Polynomials $F, H_1, \ldots, H_{s+1}$ in $K[x]$ and integers $w \geqslant 0$, $j \geqslant 0$, $n \geqslant 1$, such that:
- for all $k \in \{1, \ldots, s\}$, H_k is monic of degree $d_k = \deg H_k$ and has w-valuation $j_k = wd_k$,
- H_{s+1} has degree at most $d_{s+1} := \deg F - d_1 - \cdots - d_s$ and has w-valuation $j_{s+1} := j - j_1 - \cdots - j_s$,
- $[F]_{j,w} = [H_1 \cdots H_{s+1}]_{j,w}$,
- For all $k_1 \neq k_2$, the resultant $\mathrm{Res}(H_{k_1}, H_{k_2})$ has valuation $d_{k_1} j_{k_2}$.

Output $H_1^*, \ldots, H_{s+1}^*$ in $K[x]$ such that:
- for all $k \in \{1, \ldots, s\}$, H_k^* is monic of degree d_k and $[H_k^*]_{j_k,w} = [H_k]_{j_k,w}$,
- $[F]_{0\ldots j+n,w} = [H_1^* \cdots H_{s+1}^*]_{0\ldots j+n,w}$.

1. If $s = 0$ then return $H_1^* := F$.
2. Let $h := \lfloor (s+1)/2 \rfloor$.
3. Compute $G_1 := H_1 \cdots H_h$, and $G_2 := H_{h+1} \cdots H_{s+1}$, $g_1 := j_1 + \cdots + j_h$, and $g_2 := j_{h+1} + \cdots + j_{s+1}$.
4. Call Algorithm 3 with input F, G_1, G_2, w, g_1 and n and let G_1^* and G_2^* denote the output.
5. Call Algorithm 4 recursively with input $G_1^*, H_1, \ldots, H_h, w, g_1, n$ and let $H_1^*, \ldots, H_h^*$ be the output.
6. Call Algorithm 4 recursively with input G_2^*, $H_{h+1}, \ldots, H_{s+1}$, w, g_2, n and let $H_{h+1}^*, \ldots, H_{s+1}^*$ be the output.

Proposition 5 *Algorithm 4 works correctly as specified.*

Proof The proof follows from induction on s *via* Proposition 4. □

Lemma 13 *Assume that R has dimension $r = 1$, and let F be a polynomial in $R[x]$ of degree d. Then Algorithm 4 can run so that it performs $O(\mathsf{M}(d)\log d \log s)$ operations in κ, and $O(\mathsf{M}(d)\log s)$ operations in $R/\mathfrak{m}^l$, for each value of l in $\{1, 2, 4, \ldots, 2^\lambda | 2^\lambda < n\} \cup \{n\}$.*

Proof The proof follows from induction on s *via* Lemma 12. □

3.3 Local solver with splitting

In order to decrease the cost of the shifts in Algorithm 1, we modify step 3 as follows:

Algorithm 5

Input A polynomial $F \in O_0[x]$, $w \in \mathbb{N}$, $i \in \mathbb{N}$, $m \in \mathbb{N}$, $c \in K_{i-(w-1)m}$ and $n \in \mathbb{N}$, such that $i = \nu_{w-1}(F) \leqslant n-1$, $m = \mathrm{mult}(0, [F]_{i,w-1}) \geqslant 1$, and c is the coefficient of degree m in $[F]_{i,w-1}$.

Output A set of at most m disjoint classes representing the roots of F in K with valuation at least w and to precision n.

1. Search for the first nonzero w-homogeneous component $\tilde{H}$ of F taken modulo x^m, of w-valuation k, with $i+1 \leqslant k \leqslant \min(i+m-1, n-1)$.
   a. If such a component $\tilde{H}$ does exist then
      set $j := k$ and $H := \tilde{H} = [F]_{j,w}$
      else
      if $i+m \leqslant n-1$ then set $j := i+m$, $\tilde{H} := [F]_{j,w} \bmod x^m$, and $H := \tilde{H} + cx^m = [F]_{j,w}$, otherwise return $\{O_w\}$.
   b. If H has degree 0 then return $\{\}$.
2. Compute all the roots $z_1, \ldots, z_s$ in K_w of H to precision $j+1$, together with their respective multiplicities $m_1, \ldots, m_s$.
3. a. By means of Algorithm 4, compute the factorization of F into $H^*_{s+1} \prod_{e=1}^{s} H^*_e$, where $[H^*_e]_{wm_e,w}(x) = [(x-z_e)^{m_e}]_{wm_e,w}$ for $e \in \{1, \ldots, s\}$.
   b. For each e in $1, \ldots, s$ do
      i. If $m_e = m$ then let $c_e := c$, and $F_e := F(z_e + x)$ to precision n.
      Otherwise compute $h_e := \prod_{f=1, f\neq e}^{s+1} H^*_f(z_e)$ and let $F_e := h_e H^*_e(z_e + x)$ to precision n, and $c_e := [h_e]_{j-wm_e}$.
      ii. Call Algorithm 5 recursively with entries F_e, $w+1$, j, m_e, c_e and n, in order to obtain the set $Z_{w+1,z}$ representing the roots of F_e of valuation at least $w+1$ to precision n.
4. Return $\{z + z' | z \in Z_w, z' \in Z_{w+1,z}\}$.

Proposition 6 *Algorithm 5 works correctly as specified.*

Proof The algorithm exits at step 1.a with $\{O_w\}$ whenever $\nu_w(F) \geqslant n$, which is correct. It exits at step 1.b with the empty set whenever H is a constant, which is also correct since $H = [F]_{j,w}$ by Lemma 3.

Then the proof is done by descending induction on w. If $w \geqslant n$ then the algorithm necessarily exits at step 1. Let us now assume that the proposition holds for $w+1 \leqslant n$. By Lemma 3 again we have that $H = [F]_{j,w}$. In step 3.b, if $m_e = m$, then Lemma 3 guarantees that c is actually the coefficient of degree m in $[F]_{j,w}$, and thus of $[F_e]_{j,w}$.

Assume that $m_e \neq m$. By construction, $\nu(h_e) = \sum_{f\neq e} \nu(H^*_f(z_e + b)) = j - wm_e$, for all $b \in O_{w+1}$. Therefore an element $b \in O_{w+1}$ is a root of $F(z_e + x)$ to precision n, if, and only if, b is a root of F_e to precision n. The correctness thus follows from Lemma 4. □

Algorithm 5 behaves in the same way as Algorithm 1 regarding to the nature of the recursive calls, to the intermediate values taken by w, i, m, c, and to the successive outputs, as exemplified by running it on the input considered in Example 4:

Example 6 With $R = \mathbb{Q}[[t]]$, here is the trace of Algorithm 5 with input $F(x) = x^3 - (1+t)x^2 + t^3$, $w = 0$, $i = -3$, $m = 3$, $c = 1$, and $n = 4$:

1. $j = 0$ and $H(x) = x^3 - x^2$.
2. $z_1 = 0$, $m_1 = 2$, $z_2 = 1$, $m_2 = 1$.
3. a. Hensel lifting is called with input $F(x)$, $H_1(x) := x^2$, $H_2(x) := x - 1$, $H_3(x) := 1$, $w = 0$, $j = 0$ and $n = 4$. In return we obtain $H_1^*(x) = x^2 - t^3 x - t^3$ and $H_2^*(x) = x - 1 - t + t^3$.
   b. Algorithm 1 is called recursively with input $F_1(x) = (-1-t+t^3)H_1^* = (-1-t+t^3)x^2 + t^3 x + t^3$, $w = 1$, $i = 0$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
      1. $j = 2$ and $H(x) = -x^2$.
      2. $z_1 = 0$, $m_1 = 2$.
      3. Algorithm 1 is called recursively with input $F_1(0+x)$, $w = 2$, $i = 2$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
         1. $j = 3$, $H(x) = t^3$, and the algorithm returns $\{\}$.
      4. The algorithm returns $\{\}$.

      Algorithm 1 is then called recursively with input $F_2 = H_2^*(1+x) = x - t + t^3$, $w = 1$, $i = 0$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
      1. $j = 1$ and $H(x) = x - t$.
      2. $z_1 = t$, $m_1 = 1$.
      3. Algorithm 1 is called recursively with input $F_2(t+x) = x + t^3$, $w = 2$, $i = 1$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
         1. $j = 2$ and $H(x) = x$.
         2. $z_1 = 0$, $m_1 = 1$.
         3. Algorithm 1 is called recursively with input $F_2(t+x)$, $w = 3$, $i = 2$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
            1. $j = 3$ and $H(x) = x + t^3$.
            2. $z_1 = -t^3$, $m_1 = 1$.
            3. Algorithm 1 is called recursively with input $F_2(t - t^3 + x) = x$, $w = 4$, $i = 3$, $c = 1$, and $n = 4$, and runs as follows:
               1. The algorithm returns $\{O_4\}$.
            4. The algorithm returns $\{-t^3 + O_4\}$.
         4. The algorithm returns $\{t - t^3 + O_4\}$.
4. The algorithm finally returns $\{1 + t - t^3 + O_4\}$.

3.4 Total cost of Algorithm 5

Within the same spirit as for Theorem 1, we summarize the cost of Algorithm 5 as follows:

Theorem 2 *For any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing primitive parts and separable decompositions of polynomials in $\kappa[t_1, \ldots, t_{r-1}][x]$ of degrees at most d in x and total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$, and whose degree sum is at most nd,*

- *computing roots in $\kappa[t_1,\dots,t_{r-1}]$ of at most nd primitive polynomials of degrees* 1 *and total degrees at most $n-1$ in $t_1,\dots,t_{r-1}$,*
- *computing roots in $\kappa[t_1,\dots,t_{r-1}]$ of separable polynomials in $\kappa[t_1,\dots,t_{r-1}][x]$ of degrees at least* 2 *and at most d, of total degrees at most $n-1$ in $t_1,\dots,t_{r-1}$, and whose degree sum is at most $2(d-1)$,*
- *extracting iterated pth roots of at most $O(nd/p)$ elements in $\kappa[t_1,\dots,t_{r-1}]$,*
- *multifactor Hensel lifting of polynomials in $R[x]$ of degrees at most d, whose degree sum is at most nd, and to precision n,*
- *$O(n\mathsf{M}(d)\log^2 d)$ operations in R to precision n,*
- *shifts of polynomials in $R[x]$ of degrees at most d, whose degree sum is at most nd, and to precision n, and*
- *an additional number of $O(d)$ extractions of homogenous components of valuation v, and zero tests in each R_v, for each $v \in \{0,\dots,n-1\}$.*

Proof As in the proof of Theorem 1, we claim that running Algorithm 5 with input $F \in R[x]$ and finding the only roots in R_w instead of in K_w in step 2 actually leads to the set of roots in R of valuation at least w and to precision n. This claim can be easily proved by induction thanks to Corollary 5 that ensures that all the F_e in step 3 actually belong to $R[x]$.

We enter this modified Algorithm 5 with input F, $w=0$, $i=\nu_{-1}(F)$, $m=\mathrm{mult}(0,[F]_{i,-1})$, n, and the coefficient of degree m of $[F]_{i,-1}$. Determining the values of i and m takes no more than $O(d)$ extractions of homogenous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0,\dots,n-1\}$. The computations performed in steps 1 and 4 of Algorithms 1 and 5 are very similar: the successive quantities w, j and n are the same. Therefore the cumulative costs of steps 1 and 4 drops into $O(d)$ extractions of homogenous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0,\dots,n-1\}$.

The polynomials H occurring in step 2 of Algorithm 5 are the same of those of Algorithm 1. The cumulative cost of step 2 is thus the same as in the proof of Theorem 1.

Steps 3.a perform multifactor Hensel lifting of polynomials of degree at most m and whose degree sum does not exceed mn by Lemma 6. The same analysis holds for the total cost of the shifts. Finally, the cost for computing all the h_e in steps 3 follows from Lemma 14 below. □

Lemma 14 *Let A be a commutative ring with unity, let $F_1,\dots,F_s$ be non-constant polynomials in $A[x]$ whose sum of degrees is at most d, and let $a_1,\dots,a_s$ be in A. Then the computation $\prod_{f=1,f\neq e}^{s} F_f(a_e)$ for $e \in \{1,\dots,s\}$ can be done with $O(\mathsf{M}(d)\log^2 d)$ operations in A.*

Proof In order to perform the computation we appeal to the classical divide-and-conquer paradigm:

1. Let $h := \lfloor s/2 \rfloor$. We recursively compute $\prod_{f=1,f\neq e}^{h} F_f(a_e)$ for $e \in \{1,\dots,h\}$ and then $\prod_{f=h+1,f\neq e}^{s} F_f(a_e)$ for $e \in \{h+1,\dots,s\}$.
2. We compute $G_1 := F_1 \cdots F_h$ and $G_2 := F_{h+1} \cdots F_s$ with $O(\mathsf{M}(d)\log s)$ operations in A by [22, Lemma 10.4].

3. We compute $G_1(a_{h+1}),\dots,G_1(a_s)$ and $G_2(a_1),\dots,G_2(a_h)$ with $O(\mathsf{M}(d)\log d)$ operations in A by [22, Theorem 10.6].
4. We compute the product $\prod_{f=1,f\neq e}^{s} F_f(a_e)$ as $G_2(a_e)\prod_{f=1,f\neq e}^{h} F_f(a_e)$ if $e \leqslant h$ and as $G_1(a_e)\prod_{f=h+1,f\neq e}^{s} F_f(a_e)$ otherwise.

The cost function $\mathscr{E}_A(d)$ of this algorithm thus satisfies

$$\mathscr{E}_A(d) \in \mathscr{E}_A(\deg G_1) + \mathscr{E}_A(\deg G_2) + O(\mathsf{M}(d)\log d).$$

We deduce that $\mathscr{E}_A(d) \in O(\mathsf{M}(d)\log^2 d)$. □

As for Algorithm 1, we focus on the case of dimension 1. Remark that in dimension 1 the computation of the h_e in step 3 of Algorithm 5 can be discarded. In fact it suffices to take $h_e := t_r^{j-wm_e}$. The purpose of the h_e is only to ensure that the F_e actually belong to $R[x]$ whenever $r \geqslant 2$.

Corollary 6 *Let $\mathbb{K}$ be a field, and let R be the power series ring $\mathbb{K}[[t]]$. Then, for any polynomial F in $R[x]$ of degree at most d and given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in $\mathbb{K}$ of separable polynomials in $\mathbb{K}[[x]]$ of degrees at least 2, and whose degree sum is at most $2(d-1)$,*
- *extracting iterated pth roots of at most $O(nd/p)$ elements in $\mathbb{K}$, and*
- *an additional number of $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ arithmetic operations in $\mathbb{K}$.*

Proof This is a corollary of Theorem 2. By [36, Proposition 5], the cumulative cost of the separable factorizations amounts to $O(n\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. The cumulative cost of the shifts in steps 3 is in $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 7. Finally, the cumulative cost of the Hensel liftings in steps 3 is also in $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 12. □

Corollary 7 *Let $\mathbb{K}$ be a field of characteristic 0 and let R be the power series ring $\mathbb{K}[[t]]$. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in $\mathbb{K}$ of separable polynomials whose degree sum is at most $2(d-1)$, and*
- *an additional number of $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ arithmetic operations in $\mathbb{K}$.*

Proof This follows directly from the previous corollary. □

Corollary 8 *Let R be the power series ring $\mathbb{F}_q[[t]]$ over the finite field with $q = p^k$ elements. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of*

$$O\left((n\mathsf{M}(n)+\log(dq))\mathsf{M}(d)\log d + n\frac{d}{p}\log(q/p)\right)$$

operations in $\mathbb{F}_q$.

Proof By [22, Corollary 14.16] and Corollary 6, the cumulative cost for root-finding amounts to $O(\mathsf{M}(d)\log d\log(dq))$ operations in $\mathbb{F}_q$. □

Corollary 9 *Let R be an unramified extension of $\mathbb{Z}_p$ of degree k. Then, for any given polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of $\tilde{O}((n+k\log p)ndk\log p)$ bit-operations.*

Proof This is again a corollary of Theorem 2. In fact, by [36, Proposition 5], the cumulative cost of the primitive parts and separable factorizations amounts to $\tilde{O}(nd)$ operations in $\mathbb{F}_q$, where $q := p^k$, which boils down to $\tilde{O}(ndk\log p)$ bit-operations. By [22, Corollary 14.16], the cumulative cost for root-finding amounts to

$$O(\mathsf{M}(d)\log d\log(dq))$$

operations in $\mathbb{F}_q$, whence $\tilde{O}(d(k\log p)^2)$ bit-operations. The iterated root extractions take $O\left(\frac{nd}{p}\log\frac{q}{p}\right)$ operations in $\mathbb{F}_q$. Finally, the cumulative cost of the shifts and Hensel liftings in steps 3 is in $\tilde{O}(n^2dk\log p)$. □

3.5 Implementation and timings

In this subsection we compare the performances of Algorithms 1 and 5 for computing all the roots of polynomials F in $\mathbb{Z}/p^n\mathbb{Z}$, where $p := 73$. The family of polynomials F we have taken depends on the parameter d for the degree, n for the precision, and s for the number of roots. In fact F is built as the product of s random monic linear factors times a random polynomial of degree $d-s$.

Our implementation uses the C++ library of MATHEMAGIX [29]. It is freely available in the QUINTIX package from the SVN server of MATHEMAGIX at `http://gforge.inria.fr/projects/mmx/`.
For the present examples, the root finding for $\mathbb{Z}/p\mathbb{Z}[x]$ uses a naive exhaustive search, which turns out to be very fast whenever p is sufficiently small. Product of polynomials in $\mathbb{Z}/p^n\mathbb{Z}[x]$ is performed *via* the Kronecker substitution [22, Chapter 8, Section 4] which reduces to multiplying large integers with GMP [24]. For all the timings we used one core of an Intel(R) Xeon(R) CPU E5520 at 2.27 GHz with 72 Gb of memory, and display timings in milliseconds.

In Tables 1 and 3 we report on the time spent by Algorithm 1 for various values of d, n and s. Tables 2 and 4 concern the same computations but performed by Algorithm 5. As expected performances of Algorithm 1 behave roughly quadratically in d, while the ones of Algorithm 5 are roughly linear in d, hence much higher. In these computations we could observe that most of the time of Algorithm 1 is spent in the shifts, while most of the time of Algorithm 5 is spent in Hensel lifting. Notice that when s becomes large in Table 1, the multiplicities of more and more roots of step 2 of Algorithm 1 become greater than the precision n, which leads to less recursive calls hence to a total cost less than expected.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	4	17	78	380	1623	5802	8527
$s := \lfloor \sqrt{d} \rfloor$	2	5	17	65	242	878	3290

Table 1 Algorithm 1 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 10$.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	4	8	18	38	82	178	373
$s := \lfloor \sqrt{d} \rfloor$	2	3	6	12	24	55	113

Table 2 Algorithm 5 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 10$.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	409	2191	12212	68944	358565	2120061	10754404
$s := \lfloor \sqrt{d} \rfloor$	166	671	2512	10635	42700	175846	657423

Table 3 Algorithm 1 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 100$.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	229	474	984	2085	4431	9615	21135
$s := \lfloor \sqrt{d} \rfloor$	95	151	228	390	676	1346	2616

Table 4 Algorithm 5 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 100$.

3.6 Cost analysis in higher dimension

When R has dimension $r \geqslant 2$, the naive algorithm has the advantage to operate directly in R, while Algorithm 5 needs to perform divisions in K, which has the drawback to cause an expression swell in the lifting stage. In this subsection we propose a probabilistic approach to avoid this expression swell.

If $a = (a_1, \ldots, a_{r-1})$ is a point in κ^{r-1}, then we write τ_a for the homomorphism from R into R that sends t_i to $(a_i + t_i)t_r$ for all $i \in \{1, \ldots, r-1\}$. If $H(x) = \sum_{l=0}^{d} H_l x^l$ is a polynomial in $R[x]$ then we further set $\tau_a(H)(x) := \sum_{l=0}^{d} \tau_a(H_l)x^l$. Remark that the image of an homogeneous element $c = \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = \nu(c)} c_e t_1^{e_1} \cdots t_r^{e_r}$ in R by τ_a is

$$\tau_a(c) = \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = \nu(c)} c_e (a_1 + t_1)^{e_1} \cdots (a_{r-1} + t_{r-1})^{e_{r-1}} t_r^{\nu(c)}.$$

Therefore c can be recovered from its value $\tau_a(c)$ if the latter is known to precision $\nu(c)+1$ in t_r and modulo $(t_1, \ldots, t_{r-1})^{\nu(c)+1}$. More generally, if c is any element of R, and if we are given $\tau_a(c)$ to precision $l+1$ in t_r and modulo $(t_1, \ldots, t_{r-1})^{l+1}$, then we can recover c modulo $(t_1, \ldots, t_r)^{l+1}$.

Following the discussion on R at the beginning of this article (based on [15, Theorem 15]), if R is the power series ring $\kappa[[t_1, \ldots, t_r]]$ then we let

$$S := \mathrm{Quot}(R/(t_r)) \otimes_{\kappa[[t_r]]} R = \kappa((t_1, \ldots, t_r))[[t_r]].$$

Otherwise, if $R = D[[t_1, \ldots, t_{r-1}]]$, where D is a *complete discrete valuation ring* with maximal ideal generated by $p = t_r$ and residue field $\kappa = D/(p)$, then we let

$$S := \mathrm{Quot}(R/(p)) \otimes_D R = D((t_1, \ldots, t_{r-1})).$$

In both cases, S is a complete commutative Noetherian unramified regular local domain of dimension 1 with maximal ideal $\mathfrak{n} = (t_r)$. We can therefore apply our algorithms in S instead of R as follows:

Lemma 15 *For any input of Algorithm 3, there exists a nonzero polynomial A in $\kappa[x_1,\ldots,x_{r-1}]$ of degree $d_1 j_2 = w d_1 d_2$ such that, for any point $(a_1,\ldots,a_{r-1}) \in \kappa^{r-1}$ satisfying $A(a_1,\ldots,a_{r-1}) \neq 0$, Algorithm 3 can run on $\tau_a(F)$, $\tau_a(H_1)$, $\tau_a(H_2)$ seen as in $S[x]$, w, j, and n, and returns $\tau_a(H_1^*)$, $\tau_a(H_2^*)$.*

Proof From the assumptions, one has $\rho := [\mathrm{Res}(H_1,H_2)]_{d_1 j_2}$ is nonzero. On the one hand, from the specialization property of the resultant, $[\tau_a(\rho)]_{d_1 j_2}$ equals $[\mathrm{Res}(\tau_a(H_1),\tau_a(H_2))]_{d_1 j_2}$. On the other hand, if

$$\rho = \sum_{e\in\mathbb{N}^r, e_1+\cdots+e_r=d_1 j_2} \rho_e t_1^{e_1}\cdots t_r^{e_r},$$

then $[\tau_a(\rho)]_{d_1 j_2} = \sum_{e\in\mathbb{N}^r, e_1+\cdots+e_r=d_1 j} \rho_e a_1^{e_1}\cdots a_{r-1}^{e_{r-1}} t_r^{d_1 j}$. We thus let

$$A(x_1,\ldots,x_{r-1}) := \sum_{e\in\mathbb{N}^r, e_1+\cdots+e_r=d_1 j_2} \rho_e x_1^{e_1}\cdots x_{r-1}^{e_{r-1}}.$$

If $A(a_1,\ldots,a_{r-1}) \neq 0$ then $\tau_a(F)$, $\tau_a(H_1)$, $\tau_a(H_2)$, w, j and n satisfy the requirements of Algorithm 3. □

Lemma 16 *For any input of Algorithm 4, there exists a nonzero polynomial A in $\kappa[x_1,\ldots,x_{r-1}]$ of degree at most $w(\deg F)^2/2$ such that, for any point $(a_1,\ldots,a_{r-1}) \in \kappa^{r-1}$ satisfying $A(a_1,\ldots,a_{r-1}) \neq 0$, Algorithm 4 can run on $\tau_a(F)$, $\tau_a(H_1),\ldots,\tau_a(H_{s+1})$, seen as in $S[x]$, w, j, and n, and returns $\tau_a(H_1^*),\ldots,\tau_a(H_{s+1}^*)$.*

Proof Let $A_{i,j}$ be the polynomial A of Lemma 15 applied to $H_i H_j, H_i, H_j$, for $i<j$. By the multiplicativity of the resultant it suffices to take $A := \prod_{i<j} A_{i,j}$. The degree of A is $w\sum_{i<j} d_i d_j$, according to the notation of Algorithm 4. The latter sum is bounded by $w\deg(F)^2/2$. □

In order to apply Algorithm 5, it suffices to pick up at random a point $(a_1,\ldots,a_{r-1}) \in \kappa^{r-1}$, then to perform the Hensel lifting to precision n in t_r and modulo $(t_1,\ldots,t_{r-1})^n$, to compute $\tau_a(F_e)$, and finally to recover F_e in $R[x]$ since it actually belongs to $R[x]$. In this way, if κ has sufficiently many elements, then Algorithm 5 behaves efficiently in high dimension with a high probability of success.

4 Application to error correcting codes

Let E be an unramified extension of $\mathbb{Z}_p$ of degree k so that $E/(p^n)$ is the Galois ring $\mathrm{GR}(p^n,k)$ of Definition 1, and let $q := p^k$.

4.1 Algorithm

Let F be a polynomial in $E[t][x]$ of degree at most d in x and degree at most d_t in t. We are interested in computing all the roots of F in $E[t]$ of degree at most a given integer l, and modulo p^n.

Algorithm 6

Input A polynomial $F \in E[t][x]$ of degree at most d in x and d_t in t, and two non-negative integers n and l.

Output A set of at most d disjoint classes representing the roots of F in $E[t]$ of degree at most l modulo p^n.

1. Compute an irreducible polynomial $\varphi(t) \in \mathbb{F}_q[t]$ of degree $e = dl + d_t + 1$.
2. Call Algorithm 1 or 5 with $R := E[t]/(\varphi(t))$, and F seen in $R[x]$ of degree at most d, in order to obtain a set Z of at most d disjoint classes of the roots.
3. Return the elements of Z of degree at most l in t.

Proposition 7 *Algorithm 6 works correctly, and takes:*

- *an expected number of* $\tilde{O}((n^2 d + \max(1, n/p)ek \log p)dek \log p)$ *bit-operations when using the naive solver derived from Algorithm 1, or*
- *an expected number of* $\tilde{O}((n^2 + nek \log p)dek \log p)$ *bit-operations when using the solver derived from Algorithm 5.*

Proof A polynomial $z(t)$ is a root of F of degree at most l modulo p^n if, and only if, it is a root of F seen in $E[t]/(\varphi(t))$ modulo p^n, since $F(z(t))$ has degree at most $dl + d_t$.

Step 1 can be done with an expected number of $\tilde{O}(e^2 \log q)$ operations in $\mathbb{F}_q$ by [22, Corollary 14.43]. The cost of step 2 then follows from Corollary 4 (resp. from Corollary 9) when using Algorithm 1 (resp. using Algorithm 5). □

4.2 Experiments

We have implemented finite fields in the C++ package of MATHEMAGIX called FINITEFIELDZ. Several representations and algorithms are available, including products *via* lookup tables for small fields, a wrapper of the MPFQ library [23] for specific fields, and a generic implementation as quotient ring for larger fields. We have also implemented Galois rings in the aforementioned QUINTIX package, in a way very similar to finite fields. Root finding can be performed either by an exhaustive search or *via* Berlekamp or Cantor-Zassenhaus based algorithms (see for instance [22, Chapter 14]).

Algorithm 6 is available in the QUINTIX package. In order to test it, we built input polynomials from real examples by using Sudan's interpolation algorithm for Reed-Solomon codes over Galois rings [43, Lemma 4]. This interpolation relies merely on linear algebra over Galois rings as described in [2,3]. In Tables 5 and 6 we display the performances of Algorithm 6 for various length of the code. Timings are measured in milliseconds in the same conditions as in Section 3.5, and we compare the relative performances of Algorithms 1 and 5.

Length of the code	100	200	250
p	103	211	257
d	3	2	2
d_t	29	116	83
l	9	49	59
e	57	215	202
Algorithm 1 in step 2 (ms)	785	5492	3509
Algorithm 5 in step 2 (ms)	1068	10298	14978

Table 5 Algorithm 6 for Reed-Solomon codes over $\mathbb{Z}/p^{10}\mathbb{Z}$.

Length of the code	100	200	250
p	103	211	257
d	3	2	2
d_t	29	116	83
l	9	49	59
e	57	215	202
Algorithm 1 in step 2 (ms)	675	11421	6134
Algorithm 5 in step 2 (ms)	2046	9942	10861

Table 6 Algorithm 6 for Reed-Solomon codes over $\mathbb{Z}/p^{100}\mathbb{Z}$.

Length of the code	1400	1800	2000
p	1409	1811	2003
d	6	8	10
d_t	43	50	45
l	9	9	9
e	98	123	136
Algorithm 1 in step 2 (ms)	19742	58414	145380
Algorithm 5 in step 2 (ms)	23818	70941	140828

Table 7 Algorithm 6 for Reed-Solomon codes over $\mathbb{Z}/p^{10}\mathbb{Z}$ with a forced degree d for the interpolation polynomial F.

Notice that the timings are somehow similar between precision 10 and 100. This is mainly because the interpolation step returns a polynomial whose coefficients have valuations close to the precision. Moreover the degrees in x being very small compared to the extension degree of the Galois ring used by Algorithm 6 in step 2, both Algorithms 1 and 5 spend a lot of time in the root-finding algorithm over large finite fields.

In the latter examples, we can see that the degree d is rather small in comparison to d_t. Heuristically, this fact could be related to [40, Proposition 12, page 9] which states that the probability of having more than one codeword in a Hamming ball, whose radius corresponds to the Sudan algorithm decoding radius, is close to zero. The degree d of F is related to the number c of codewords within the Hamming ball by $c \leqslant d$. And, in practice, we observe that d is close to 1 when $c = 1$ with probability close to 1.

Of course one can construct received words such that the decoding algorithm has to return a given number c of codewords. Hence, by the inequality $c \leqslant d$, one can force the degree d to be at least a given positive integer. Such a word can be built as follows. First denote by $(r)_{i\cdots j}$ the vector $(r_i, r_{i+1}, \ldots, r_j)$ for any vector r

with coefficients in $\mathbb{Z}/p^n\mathbb{Z}$. Take d codewords $c_1,\ldots,c_d$ such that $(c_1)_{1\cdots k-d-1} = (c_2)_{1\cdots k-d-1} = \cdots = (c_d)_{1\cdots k-d-1}$ where k is the rank of the code. Then compute $\Delta = \lfloor (\ell-k-d)/d \rfloor$ where ℓ is the length of the code. Finally compute the word

$$\rho = ((c_1)_{1\cdots k-d-1}; (c_1)_{k-d\cdots k-d+\Delta}; (c_2)_{k-d+\Delta+1\cdots k-d+2\Delta}; \cdots; (c_d)_{k-d+(d-1)\Delta\cdots k-d+d\Delta}),$$

and truncate ρ, if necessary, so that its length equals the length ℓ of the code. Table 7 reports on timings obtained with this construction. Notice that Algorithm 5 starts to be interesting when the degree d is at least 10 for codes with a very low rate. In this case the code rate is smaller than 0.5%. Therefore the naive algorithm turns out to be sufficient for practical applications whenever the code rate is close to 1.

Acknowledgements This work has been partly supported by the French ANR-09-JCJC-0098-01 MAGIX project, and by the DIGITEO 2009-36HD grant of the Région Île-de-France. We would like to thank DANIEL AUGOT and both referees for their useful comments on this article.

References

1. Alekhnovich, M.: Linear Diophantine equations over polynomials and soft decoding of Reed-Solomon codes. IEEE Trans. Inform. Theory **51**(7), 2257–2265 (2005)
2. Armand, M.A.: Improved list decoding of generalized Reed-Solomon and alternant codes over rings. In: IEEE International Symposium on Information Theory 2004 (ISIT 2004), p. 384 (2004)
3. Armand, M.A.: List decoding of generalized Reed-Solomon codes over commutative rings. IEEE Trans. Inform. Theory **51**(1), 411–419 (2005)
4. Augot, D., Barbier, M., Couvreur, A.: List-decoding of binary Goppa codes up to the binary Johnson bound. In: Information Theory Workshop (ITW), 2011 IEEE, pp. 229–233 (2011)
5. Augot, D., Zeh, A.: On the Roth and Ruckenstein equations for the Guruswami-Sudan algorithm. In: IEEE International Symposium on Information Theory - ISIT 2008, pp. 2620–2624. IEEE, Toronto, Canada (2008)
6. Bartley, K.G.: Decoding algorithms for algebraic geometric codes over rings. Ph.D. thesis, University of Nebraska (2006)
7. Berlekamp, E.R.: Algebraic coding theory. M-6. Aegean Park Press (1984)
8. Berlekamp, E.R., Welch, L.R.: Error correction for algebraic block codes (1986). Patent 4633470
9. Berthomieu, J., van der Hoeven, J., Lecerf, G.: Relaxed algorithms for p-adic numbers. J. Théor. Nombres Bordeaux **23**(3) (2011)
10. Bini, D., Pan, V.Y.: Polynomial and matrix computations. Vol. 1. Fundamental algorithms. Progress in Theoretical Computer Science. Birkhäuser (1994)
11. Bostan, A., Schost, É.: Polynomial evaluation and interpolation on special sets of points. J. Complexity **21**(4), 420–446 (2005)
12. Cantor, D.G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Inform. **28**, 693–701 (1991)
13. Chudnovsky, D.V., Chudnovsky, G.V.: On expansion of algebraic functions in power and Puiseux series. I. J. Complexity **2**(4), 271–294 (1986)
14. Chudnovsky, D.V., Chudnovsky, G.V.: On expansion of algebraic functions in power and Puiseux series. II. J. Complexity **3**(1), 1–25 (1987)
15. Cohen, I.S.: On the structure and ideal theory of complete local rings. Trans. Amer. Math. Soc. **59**, 54–106 (1946)
16. Duval, D.: Rational Puiseux expansions. Compositio Math. **70**(2), 119–154 (1989)
17. Fröhlich, A., Shepherdson, J.C.: Effective procedures in field theory. Philos. Trans. Roy. Soc. London. Ser. A. **248**, 407–432 (1956)
18. Fürer, M.: Faster integer multiplication. In: Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing (STOC 2007), pp. 57–66. ACM (2007)

19. Gao, S.: A new algorithm for decoding Reed-Solomon codes. In: V. Bhargava, H.V. Poor, V. Tarokh, S. Yoon (eds.) Communications, Information and Network Security, *The Springer International Series in Engineering and Computer Science*, vol. 712, pp. 55–68. Springer US (2003)
20. Gao, S., Shrokrollahi, M.A.: Computing roots of polynomials over function fields of curves. In: D. Joyner (ed.) Coding theory and cryptography: from enigma and Geheimschreiber to quantum theory, pp. 214–228. Springer Berlin Heidelberg (2000)
21. von zur Gathen, J.: Hensel and Newton methods in valuation rings. Math. Comp. **42**(166), 637–661 (1984)
22. von zur Gathen, J., Gerhard, J.: Modern computer algebra, second edn. Cambridge University Press (2003)
23. Gaudry, P., Thomé, E.: MPFQ, a finite field library. Available at `http://mpfq.gforge.inria.fr` (2008)
24. Granlund, T., et al.: GMP, the GNU multiple precision arithmetic library. Available at `http://gmplib.org` (1991)
25. Griffiths, D.: Series expansions of algebraic functions. In: W. Bosma, A. Poorten (eds.) Computational Algebra and Number Theory, *Mathematics and Its Applications*, vol. 325, pp. 267–277. Springer Netherlands (1995)
26. Guruswami, V., Sudan, M.: Improved decoding of Reed-Solomon and algebraic-geometric codes. IEEE Trans. Inform. Theory **45**, 1757–1767 (1998)
27. Hallouin, É.: Computing local integral closures. J. Symbolic Comput. **32**(3), 211–230 (2001)
28. Henry, J.P., Merle, M.: Complexity of computation of embedded resolution of algebraic curves. In: Proceedings of Eurocal 87, *Lecture Notes in Computer Science*, vol. 378, pp. 381–390. Springer-Verlag (1987)
29. van der Hoeven, J., et al.: Mathemagix. Software available from `http://www.mathemagix.org` (2002)
30. Iwami, M.: Extension of expansion base algorithm for multivariate analytic factorization including the case of singular leading coefficient. SIGSAM Bull. **39**(4), 122–126 (2005)
31. Kedlaya, K.S.: The algebraic closure of the power series field in positive characteristic. Proc. Amer. Math. Soc. **129**(12), 3461–3470 (2001)
32. Kötter, R.: On algebraic decoding of algebraic-geometric and cycling codes. Ph.D. thesis, Linköping University, Sweden (1996)
33. Kötter, R., Vardy, A.: Algebraic soft-decision decoding of Reed-Solomon codes. IEEE Trans. Inform. Theory **49**(11), 2809–2825 (2003)
34. Kuo, T.C.: Generalized Newton-Puiseux theory and Hensel's lemma in $\mathbb{C}[[x,y]]$. Canad. J. Math. **41**(6), 1101–1116 (1989)
35. Lang, S.: Algebra, *Graduate Texts in Mathematics*, vol. 211, third edn. Springer-Verlag (2002)
36. Lecerf, G.: Fast separable factorization and applications. Appl. Algebra Engrg. Comm. Comput. **19**(2), 135–160 (2008)
37. Moon, T.K.: Error correction coding: mathematical methods and algorithms. Wiley-Interscience (2005)
38. Poteaux, A., Rybowicz, M.: Complexity bounds for the rational Newton-Puiseux algorithm over finite fields. Appl. Algebra Engrg. Comm. Comput. **22**(3), 187–217 (2011)
39. Raghavendran, R.: Finite associative rings. Compositio Math. **21**, 195–229 (1969)
40. Refslund Nielsen, R., Høholdt, T.: Decoding Reed-Solomon codes beyond half the minimum distance. In: J. Buchmann, T. Høholdt, H. Stichtenoth, H. Tapia-Recillas (eds.) Coding Theory, Cryptography and Related Areas, pp. 221–236. Springer Berlin Heidelberg (2000)
41. Roth, R.M., Ruckenstein, G.: Efficient decoding of Reed-Solomon codes beyond half the minimum distance. In: IEEE International Symposium on Information Theory 1998, p. 56 (1998)
42. Schönhage, A., Strassen, V.: Schnelle Multiplikation grosser Zahlen. Computing **7**, 281–292 (1971)
43. Sudan, M.: Decoding of Reed-Solomon codes beyond the error-correction bound. J. Complexity **13**(1), 180–193 (1997)
44. Truong, T.K., Eastman, W.L., Reed, I.S., Hsu, I.S.: Simplified procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm. IEE Proc. Comput. and Digit. Tech. **135**(6), 318–324 (1988)
45. Walker, J.L.: Algebraic geometric codes over rings. J. Pure Appl. Algebra **144**(1), 91–110 (1999)
46. Walker, R.J.: Algebraic curves. Springer-Verlag, New York (1978). Reprint of the 1950 edition
47. Walsh, P.G.: On the complexity of rational Puiseux expansions. Pacific J. Math. **188**(2), 369–387 (1999)
48. Walsh, P.G.: A polynomial-time complexity bound for the computation of the singular part of a Puiseux expansion of an algebraic function. Math. Comp. **69**(231), 1167–1182 (2000)