Sparse Polynomial Interpolation and Division in Soft-linear Time

Pascal Giorgi, Bruno Grenet, Armelle Perret Du Cray, Daniel S. Roche

Sparse Polynomial Interpolation and Division in Soft-linear Time

Pascal Giorgi
LIRMM, Univ. Montpellier, CNRS
Montpellier, France
pascal.giorgi@lirmm.fr

Bruno Grenet
LIRMM, Univ. Montpellier, CNRS
Montpellier, France
bruno.grenet@lirmm.fr

Armelle Perret du Cray
LIRMM, Univ. Montpellier, CNRS
Montpellier, France
armelle.perret-du-cray@lirmm.fr

Daniel S. Roche
United States Naval Academy
Annapolis, Maryland, U.S.A
roche@usna.edu

May 19, 2022

Abstract

Given a way to evaluate an unknown polynomial with integer coefficients, we present new algorithms to recover its nonzero coefficients and corresponding exponents. As an application, we adapt this interpolation algorithm to the problem of computing the exact quotient of two given polynomials. These methods are efficient in terms of the bit-length of the sparse representation, that is, the number of nonzero terms, the size of coefficients, the number of variables, and the logarithm of the degree. At the core of our results is a new Monte Carlo randomized algorithm to recover a polynomial $f(x)$ with integer coefficients given a way to evaluate $f(\theta) \bmod m$ for any chosen integers θ and m. This algorithm has nearly-optimal bit complexity, meaning that the total bit-length of the probes, as well as the computational running time, is softly linear (ignoring logarithmic factors) in the bit-length of the resulting sparse polynomial. To our knowledge, this is the first sparse interpolation algorithm with soft-linear bit complexity in the total output size. For polynomials with integer coefficients, the best previously known results have at least a cubic dependency on the bit-length of the exponents.

1 Introduction

Sparse and supersparse polynomials. Sparse polynomial interpolation is an important and well-studied problem in computer algebra, with numerous connections to related problems in signal processing and coding theory. In our context, the task is to determine the *sparse representation* of an unknown polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$,

which is the list of nonzero coefficients $c_1, \ldots, c_t \in \mathbb{Z}$ and corresponding exponent tuples $\mathbf{e}_1, \ldots, \mathbf{e}_t \in \mathbb{N}^n$ such that

$$f = c_1 \mathbf{x}^{\mathbf{e}_1} + c_2 \mathbf{x}^{\mathbf{e}_2} + \cdots + c_t \mathbf{x}^{\mathbf{e}_t}.$$

Here we use the convenient notation for each monomial

$$\mathbf{x}^{\mathbf{e}_i} = x_1^{e_{i,1}} x_2^{e_{i,2}} \cdots x_n^{e_{i,n}}.$$

We assume every $c_i \neq 0$ and all the $\mathbf{e}_i$'s are distinct. The number of nonzero terms in f, also known as the *sparsity*, is written as $t = \#f$. The bit size of the sparse representation of f is $t(n \log D + \log H)^*$ with D the *max degree* of f, that is the largest exponent $e_{i,j}$, and H its *height*, that is the maximum magnitude of a coefficient[†].

Any sparse interpolation algorithm requires some bounds on the unknown f (typically on the degree, size of coefficients, and possibly number of nonzero terms), as well as a way to evaluate f. The algorithm constructs a series of evaluation points, performs said evaluations, then performs some computations, possibly iterating these steps before settling on the final result.

Dense polynomial interpolation algorithms have been known for centuries and can always recover a unique result, even if the evaluation points are not chosen by the algorithm. However, methods such as Lagrange interpolation scale at least linearly with the *degree* of the unknown polynomial. Sparse polynomial algorithms, by contrast, should scale according to the number of nonzero terms, which in general can be much smaller than the degree.

In fact, the degree could be *exponentially larger* than the sparse representation. Algorithms whose cost scales with the bit-length of the exponents, i.e., the logarithm of the degree, are called *supersparse* or *lacunary* polynomial algorithms.

Sparse interpolation   Sparse interpolation has received much attention since the landmark paper by Ben-Or and Tiwari [7], which provides a deterministic algorithm of complexity polynomial in T, D, n for multivariate polynomials over $\mathbb{Z}$, given a bound on $T \geq t$ as input. This algorithm is given in the context of an unknown polynomial that a black box allows to evaluate at any point of $\mathbb{Z}$ freely chosen by the algorithm. Numerous extensions have been proposed [52, 38, 30], in particular in order to: deal with finite fields [21, 26, 32, 16, 29], avoid the bound on t by *early termination* techniques [34] or extend the problem to the case of sparse rational functions [39, 37, 12, 25]. Some algorithms require the black box model to be slightly relaxed and allow evaluations in extension rings or quotient rings [21, 41, 2, 45, 37, 16, 10, 23].

Garg and Schost [14] described the first algorithm for a generic ring whose complexity is polynomial in $\log D$ (*supersparse* interpolation). Their algorithm takes as input a *straight-line program* (SLP) rather than a black box. Hence, there is no restriction on the evaluation domain, but the evaluation cost has to be taken into account. Subsequent works have refined the complexity bounds of this algorithm when the ring

---

*Unless otherwise stated, logarithms are in base 2; We shall also use base-q logarithms for some prime q, and natural logarithms for prime-related statements.

[†]In this work, we do not consider the case of *unbalanced* bit lengths, where the differing sizes of each coefficient and exponent are considered in the complexity.

of coefficients is a finite field, the ring of integers or rational numbers [4, 5, 31, 28]. The best currently known complexity is due to Huang [27] for the interpolation of an SLP of length L on a finite field $\mathbb{F}_q$ of large characteristic in $\widetilde{O}(LT \log D \log q)$ bit operations. This complexity is however not quasi-linear in the output size due to the factor $\log D$ times $\log q$.

More details on algorithms and techniques are given in Arnold's Thesis [3] or in the survey from van der Hoeven and Lecerf [24].

In unbounded coefficient domains such as $\mathbb{Z}$, the bit size of the values involved in the evaluation and computation can grow exponentially. Working with such exponential-size integers is unrealistic and may even make the problem trivial: the unknown polynomial f can be recovered from a *single evaluation* at a point larger than any coefficient, using the q-adic expansion of the result. Hence, modular techniques are needed to get efficient algorithms [36, 23]. This motivated the definition of more general black boxes that enable to perform evaluation modulo a chosen integer m.

Definition 1.1. *A* modular black box *(MBB, for short) for a multivariate polynomial* $f \in \mathbb{Z}[x_1, \dots, x_n]$ *is a function that takes any modulus* $m \in \mathbb{N}$ *and n-tuple of evaluation points* $(\theta_1, \dots, \theta_n) \in \{0, 1, \dots, m-1\}^n$, *and produces the evaluation* $f(\theta_1, \dots, \theta_n) \bmod m$.
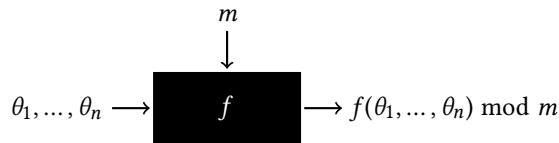
An alternative input for sparse interpolation is straight-line programs (SLP). An SLP naturally implements an MBB: Given the SLP for $f \in \mathbb{Z}[x_1, \dots, x_n]$, one can compute $f(\theta_1, \dots, \theta_n) \bmod m$. If the SLP has length L, this amounts to $O(L)$ operations in $\mathbb{Z}/m\mathbb{Z}$, or $\widetilde{O}(L(\log m + \log H))$ bit operations, where H bounds the absolute values of the constants used by the SLP. (More precisely, if the SLP uses k constants $\leq H$ in absolute value, and $H > m$, we need to reduce these k integers modulo m, in time $\widetilde{O}(k \log H)$.)

A fair analysis of a sparse interpolation algorithm over $\mathbb{Z}[\mathbf{x}]$ should therefore consider four things: (1) the number of evaluations, (2) the bit-length of these evaluations, (3) the arithmetic complexity of extra processing to produce the result, and (4) the bit-length of integers involved in the extra processing.

Sparse polynomial exact division Another issue with sparse polynomials is the complexity of the basic arithmetic operations; see the survey of Roche [48]. Even for standard operations such as multiplication or division, no deterministic quasi-linear time algorithm is known. In spite of some theoretical improvements and practical implementations, deterministic algorithms for these operations remain quadratic in the sparsity [33, 42, 43, 44, 15]. The major difficulty comes from the unpredictability of the sparsity of the result. Quite recently, new probabilistic algorithms for sparse polynomial multiplication have been proposed [6, 46, 22]. This led to the first quasi-linear algorithm for sparse polynomial multiplication over the integers or finite fields

with large characteristic [18], based on sparse interpolation and sparse polynomial verification [20].

For the Euclidean division of sparse polynomials, the case of exact division (when the remainder is known to be zero) was improved by similar techniques [19]. This led to the first algorithm that is quasi-linear in the sparsity, though not in the total bit size.

1.1 Summary of results

We provide the first truly quasi-linear sparse interpolation algorithm, for integer polynomials.

Theorem 1.2. *There is a Monte Carlo randomized algorithm that, given an MBB for an unknown polynomial $f \in \mathbb{Z}[x_1, \ldots, x_n]$ and bounds D, H, and T on respectively its max degree, height and sparsity, recovers the sparse representation of f with probability at least $\frac{2}{3}$. It requires $O(T)$ probes to the MBB plus $\widetilde{O}(T(n \log D + \log H))$ bit operations.*

Based on similar techniques, we are also able to provide the first quasi-linear time algorithm for computing the exact quotient of two sparse polynomials.

Theorem 1.3. *There is a Monte Carlo randomized algorithm that, given two sparse polynomials $f, g \in \mathbb{Z}[x_1, \ldots, x_n]$ such that g divides f and a bound T on the sparsity of the quotient f/g, computes the sparse representation of f/g with probability at least $\frac{2}{3}$. It requires $\widetilde{O}((T + \#f + \#g)(n \log D + \log H))$ bit operations where $D = \deg(f)$, and H is a bound on the height of the three polynomials f, g and f/g.*

Our algorithms are randomized of the Monte Carlo type, meaning that they can return incorrect results. By repeatedly running the algorithms and taking the majority result, the probability of error decreases exponentially in the number of iterations.

The exact division algorithm can be performed without an *a priori* sparsity bound. For that, we rely on the sparse product verification algorithm of Giorgi et al. [18, 20]. It becomes an Atlantic City algorithm (both its correctness and running time are probabilistic) since the verification algorithm is randomized of Monte Carlo type.

We present our results for multivariate polynomials but will focus on univariate polynomials in our descriptions and proofs that follow. This is allowed by the fairly classical Kronecker substitution [40, 35]. Indeed, there is a one-to-one correspondence between polynomials $f \in \mathbb{Z}[x_1, \ldots, x_n]$ with $\deg_{x_i} f < D$, and univariate polynomials in $\mathbb{Z}[x]$ of degree $< D^n$ through the transformation $f_u(x) = f(x, x^D, x^{D^2}, \ldots, x^{D^{n-1}})$. Note that Kronecker substitution preserves the bit size of the polynomials. For sparse polynomials, the transformation and its inverse require $\widetilde{O}(Tn \log D)$ bit operations. An MBB for f can simulate a univariate MBB for f_u by evaluating f at the powers of the given point. This adds a negligible cost in our algorithms since we probe the MBB on points of known low order.

The rest of the paper is then devoted to univariate polynomials. By abuse of notation we still use D to denote the degree of the univariate polynomial, instead of D^n.

1.2 Main ideas

Our new algorithms mostly combine aspects of existing techniques initiated by the work of Garg and Schost [14] and Ben-Or and Tiwari [7] plus a few new techniques. We outline the most important of them to give a broad overview of the main interpolation algorithms.

Finding candidate exponents Like in the recent line of work of Gao and Huang [27, 31, 28, 29], our overall approach is to generate *candidate* terms of the unknown sparse polynomials f. This is achieved by interpolating f mod $x^p - 1$ for *tiny* primes p, where $p \in O(T \log D)$ is so small that even performing $\widetilde{O}(p)$ operations is allowable within the targeted complexity.

This approach originates in the work of Garg and Schost [14] on SLP. In that and subsequent works, the polynomial reduced modulo $x^p - 1$ is explicitly computed using dense arithmetic. This step alone is too costly to get a quasi-linear complexity.

Our approach is to instead compute f mod $x^p - 1$ using sparse interpolation *à la* Prony. To this end, we have to evaluate f on elements of order p. If ω is the generator of an order-p subgroup of $\mathbb{F}_q$, then $f(\omega) = (f \bmod x^p - 1)(\omega)$. This allows us to recover the polynomial f modulo $\langle x^p - 1, q \rangle$. If $\mathbb{F}_q$ is a small field, namely $q \in \mathsf{poly}(p)$, this Prony-based interpolation has quasi-linear cost. Since q is rather small, this actually only provides the exponents modulo p of f, but almost no information on the coefficients.

To recover the values of the coefficients, we need to work in a ring $\mathbb{Z}/m\mathbb{Z}$ for some large modulus m. A full Prony-based sparse interpolation over that ring would be too expensive. However, the exponents of f mod $x^p - 1$ have already been computed and we only need to perform the second part of the algorithm, namely sparse interpolation with known support. Also we cannot afford to compute a large enough prime number m. Instead, we work over a prime power modulus, namely $m = q^k$ for some k. This part can still be done in quasi-linear time, even in this larger ring, since it amounts to solving structured linear system of size $O(\#f)$.

There, we can only ensure a good probability that one-half of the terms do not collide in the reduction modulo $x^p - 1$. As proposed by Huang [27] this can be easily turned into a Monte Carlo algorithm by doing $O(\log T)$ interpolations with different primes p. A second problem is that, from this step, we learn only the exponents modulo p and not the full exponents themselves. Here we can rely on the clever idea of embedding the exponents in the coefficients [23, 6, 27]. The approach of Huang [27] is to use the derivative for that purpose. This is well adapted for SLP since the derivative can be computed by means of automatic differentiation. A more general way that encompasses the MBB, reminiscent of Paillier encryption scheme [47], has been proposed by Arnold and Roche [6]. Given a modulus m, they consider both polynomials $f(x)$ and $f((1 + m)x)$ in the ring $\mathbb{Z}/m^2\mathbb{Z}$. Because of the identity $(1 + m)^{e_i} \bmod m^2 = 1 + e_i m$, the ratio of corresponding coefficients between these two polynomials reveals each exponent e_i modulo m^2, provided that term did not collide with any others. In our case, the modulus m is q^k and we actually perform the second part of the Prony-based interpolation algorithm over $\mathbb{Z}/q^{2k}\mathbb{Z}$ to compute both $f(x)$ and $f((1 + q^k)x)$ modulo $\langle x^p - 1, q^{2k} \rangle$.

Finding rings with specified subgroups   Our approach crucially relies on the ability of finding prime numbers p, q and elements ω and ω_k such that ω and ω_k are generators of order-p subgroups of respectively $\mathbb{F}_q$ and $\mathbb{Z}/q^{2k}\mathbb{Z}$. In particular, p must divide $q - 1$. Effective versions of Dirichlet's theorem on primes in arithmetic progressions tell us that, for a prime p, we can (usually) find another prime q such that $p \mid (q - 1)$, where $q \leq O(p^6)$ is not too much larger than p, see [49]. This allows us to choose q as a prime in the arithmetic progression $\{ap + 1 : a \geq 1\}$ and to set $\omega = \zeta^{(q-1)/p}$ for a random $\zeta \in \mathbb{F}_q$. Furthermore, one can easily construct an element ω_k of order p in $\mathbb{Z}/q^{2k}\mathbb{Z}$ by lifting ω through Newton iteration. We also demonstrate that ω_k is principal, which is a necessary condition to be able to solve our structured linear system which is of transposed Vandermonde type.

Notice that changing the base ring is mandatory to minimize the bit complexity. Namely, the large rings have a modulus with $O(\log D + \log H)$ bits, but we only do $\widetilde{O}(T)$ arithmetic operations in such rings. The tiny fields, by contrast, have a modulus of only $O(\log(T \log DH))$ bits, but require at most $\widetilde{O}(T \log D)$ operations.

Exact division   To compute the quotient of two sparse polynomials f and g such that g divides f, we adapt our interpolation techniques. To allow the evaluation of f/g by evaluating both f and g, we slightly change the values of p and q and ensure that ω, ω_k and their powers are not roots of g. The values of p and q do not grow too much: p remains linear in the input plus the output bit size, and q polynomial in p. Since the height and sparsity of f/g are unknown, we must discover them during the computation. The idea is to begin with small bounds for both and increase them when needed. For this we rely on sparse polynomial product and modular product verification [18, 20]. A delicate aspect is to intertwine both bound increases.

1.3   Outline of the paper

We start with a preliminary section that gives few number theoretic results that are needed to prove the correctness of our algorithms.

Section 3 provides our softly linear interpolation algorithm extending further the main idea described above. This interpolation algorithm is re-used in Section 4 to provide a similar algorithm for the computation of the exact quotient of two sparse polynomials. Moreover, we will present an unconditional algorithm that does not require any prior knowledge of the quotient, and which has an expected softly linear running time.

2   Number-theoretic preliminaries

Our algorithms use number-theoretic results that are for many of them quite standard in the sparse interpolation literature. We recall them in this section, in the specific form required for our proofs. One slightly less common routine consists in computing a primitive root of unity (PRU) of prime order p in a ring $\mathbb{Z}/q^k\mathbb{Z}$ where $q = ap + 1$ is also a prime number. We show how to use Newton iteration for this purpose.

2.1 Prime number generation

Our algorithm first computes $f \bmod x^p - 1$ where f is the polynomial to be interpolated, and p some random prime number. The goal is that not too many exponents of f collide modulo p to be able to recover the terms of f. We use a result of Arnold and Roche [6]. Note that similar results are given in other references [4, 31].

Fact 2.1 ([6, Lemma 3.4]). *Let f be a T-sparse degree-D univariate polynomial, and p be a random prime number in $(\lambda, 2\lambda)$ where $\lambda \geq \frac{5}{3\epsilon(1-\gamma)}(T-1)\ln D$ for some γ and ϵ. Then $f \bmod x^p - 1$ has at least γT collision-free terms with probability at least $1 - \epsilon$.*

To compute $f \bmod x^p - 1$, one has to evaluate f on p-PRUs. First, we need a p-PRU $\omega \in \mathbb{F}_q$ for some prime q, and then a p-PRU $\omega_k \in \mathbb{Z}/q^k\mathbb{Z}$ for some integer k. To get ω, we actually generate the triple (p, q, ω) in a single algorithm, with the required properties. In particular, we need to find two prime numbers p, q such that $p \mid (q-1)$, that is q is in the arithmetic progression $\{ap + 1 \; : \; a \geq 1\}$, and such that $q = \mathsf{poly}(p)$. To this end, we generate p at random and sample random elements $< p^6$ in the arithmetic progression until a prime q is found. Such an algorithm can be found in Arnold's Ph.D. thesis [3] with a rigorous proof based on effective versions of Dirichlet's theorem [1, 51]. The next fact presents a variant with better probability bounds and a larger range of validity. We provide the complete proof in a short note [17].

Fact 2.2. *There exists an explicit Monte Carlo algorithm which, given a bound $\lambda \geq \frac{2^{58}}{\epsilon^2}$, produces a triple (p, q, ω) that has the following properties with probability at least $1 - \epsilon$, and returns FAIL otherwise:*

- *p is uniformly distributed amongst the primes of $(\lambda, 2\lambda)$;*

- *$q \leq \lambda^6$ is a prime such that $p \mid (q-1)$;*

- *ω is a p-primitive root of unity in $\mathbb{F}_q$;*

Its worst-case bit complexity is $\mathsf{polylog}(\lambda)$. Further, if $\lambda \geq \sqrt[5]{\frac{48}{\epsilon}\ln K}$ for some integer $K > 0$, the probability that q divides K is at most ϵ.

While the rigorous proof of this fact implies to have large values for λ, it is not too difficult to see by running few experiments that such triples exist with good probability even for smaller values. One can find some preliminary experiments in our short note [17]. In this paper, we rely on Fact 2.2 to provide rigorously proven algorithm, thus implying limitations on its practicability. Nevertheless, Algorithms 2 and 3 can be turned into practical ones just by ignoring the constant $\frac{2^{58}}{\epsilon^2}$ but without any formal proof.

2.2 Generators of prime-order subgroups

In the crucial steps of our interpolation algorithm, we need to evaluate in a small size-p multiplicative subgroup within a larger ring of order q^k, where $p \mid (q-1)$ and $k \geq 1$.

In order to do so, we need a generator of the order-p subgroup of the ring $\mathbb{Z}/q^k\mathbb{Z}$, that is, a pth primitive root of unity (PRU) in the ring.

One way to obtain such a generator would be to take a random invertible element in the ring and raise it to the power $\varphi(q^k)/p = (q-1)q^{k-1}/p$ modulo q^k. The result will certainly have multiplicative order which divides p, and therefore this power of a random element is a p-PRU unless it equals 1.

Unfortunately, that approach is too costly for our purposes, because the modulus and exponent could both have roughly $k \log q$ bits. There is a solution to this: take a p-PRU ω in the field $\mathbb{Z}/q\mathbb{Z}$, and lift it to a p-PRU ω_k in $\mathbb{Z}/q^k\mathbb{Z}$ using a Newton iteration. This works because of the following elementary lemma.

Lemma 2.3. *Suppose p, q are primes such that $p \mid (q-1)$ and $k \geq 1$. Let ω_k be any p-PRU modulo q^k. Then $\omega_k \bmod q$ is also a p-PRU modulo q. Moreover, ω_k is principal, that is $\omega_k^i - 1$ is not a zero divisor for $0 < i < p$.*

Proof. Let g be any generator of $(\mathbb{Z}/q^k\mathbb{Z})^*$, which is cyclic since q^k is a prime power. Then $g \bmod q$ must also be a generator of the smaller group $(\mathbb{Z}/q\mathbb{Z})^*$; otherwise the set $\{g^i \bmod q^k\}_{i \geq 0}$ would be too small. Because g is a generator and ω_k is a p-PRU modulo q^k, we can write $\omega_k = g^{i\varphi(q^k)/p}$ for some integer $i \in \{1, 2, \ldots, p-1\}$. This means that

$$\omega_k \bmod q = g^{i\varphi(q^k)/p} \bmod q = (g \bmod q)^{i(q-1)/p} \bmod q,$$

where we use the fact that $\varphi(q^k) = (q-1)q^{k-1}$ and $a^q \bmod q = a$ for any integer a. Because $g \bmod q$ is a generator modulo q, and $1 \leq i \leq p-1$, this means that $\omega_k \bmod q$ is a p-PRU modulo q.

For the second part, since $\omega_k \bmod q$ is a p-PRU, $\omega_k^i - 1 \bmod q \neq 0$ for $0 < i < p$. And zero divisors modulo q^k must be multiple of q, since q is prime. $\qquad\square$

Roughly speaking, Lemma 2.3 states that there is a 1-1 correspondence between p-PRUs modulo q and p-PRUs modulo q^k. In particular, for any p-PRU ω modulo q, there is a unique p-PRU ω_k modulo q^k such that $\omega_k \bmod q = \omega$. We construct the larger p-PRU ω_k through a standard Newton iteration, solving the equation $\omega_k^p - 1 = 0$ modulo higher and higher powers of q. Assuming we know $\omega_i = \omega_k \bmod q^i$ already, write $\omega_{2i} = \omega_i + aq^i$, where $a < q^i$ consists of the next i base-q digits of ω_k. Solving the modular equation $\omega_{2i}^p \bmod q^{2i} = 1$ gives

$$a = \left(\frac{1 - \omega_i^p \bmod q^{2i}}{q^i} \right) \omega_i p^{-1} \bmod q^i,$$

where the fraction divided by q^i is exact integer division, and the inverse p^{-1} is modulo q^i.

Theorem 2.4. *Provided ω is a p-PRU modulo q, Algorithm 1 returns a p-PRU ω_k modulo q^k. It has bit complexity $\widetilde{O}(k \log^2 q)$.*

Proof. The loop runs $O(\log k)$ times. The dominating step is $\omega_i^p \bmod q^{2i}$ at the last phase of the Newton iteration with $2i \geq k$. Because $p < q$, this gives the stated bit complexity. $\qquad\square$

Algorithm 1: LIFTPRU

Input: Primes p, q with $p \mid (q - 1)$, a p-PRU $\omega \in \mathbb{F}_q$ and an integer $k \geq 1$
Output: ω_k, a p-PRU modulo q^k

---

1 $i \leftarrow 1$; $\omega_1 \leftarrow \omega$
2 **while** $i < k$ **do**
3 $a \leftarrow \omega_i^p \bmod q^{2i}$
4 $a' \leftarrow (1 - a)/q^i$ using exact integer division
5 $a'' \leftarrow a' \omega_i p^{-1} \bmod q^i$
6 $\omega_{2i} \leftarrow \omega_i + a'' q^i$
7 $i \leftarrow 2i$
8 **return** $\omega_i \bmod q^k$

---

3   Univariate Interpolation

In this section, we present a Monte Carlo algorithm to interpolate a sparse polynomial given through an MBB. Our algorithm builds on classical techniques but with the originality to use non-integral domains and not only finite fields. We first recall some of these techniques before describing the algorithm.

Given an MBB for f, we need to compute the exponents of $f \bmod x^p - 1$. We note that evaluating f at powers of a p-th primitive root of unity (p-PRU) ω is equivalent to evaluating $f \bmod x^p - 1$ at the same points. As in the classical Ben-Or–Tiwari algorithm, given the sequence $f(1), f(\omega), \ldots, f(\omega^{2T-1})$, we can compute a degree-$\leq T$ annihilator polynomial Λ in $\widetilde{O}(T)$ operations in $\mathbb{F}_q$ using fast Berlekamp-Massey algorithm [50, 13]. The roots of Λ are the ω^e where $e < p$ belongs to the support of $f \bmod x^p - 1$. In our case, p is small and these exponents can be retrieved in $\widetilde{O}(p)$ arithmetic operations using Bluestein's chirp transform [9] to evaluate Λ at $1, \omega, \ldots, \omega^{p-1}$. Altogether, this gives the following.

Fact 3.1. *Given the evaluations of a T-sparse polynomial $f \in \mathbb{F}_q[x]$ at $1, \omega, \ldots, \omega^{2T-1}$ where $\omega \in \mathbb{F}_q$ is a p-PRU, one can compute the exponents of $f \bmod x^p - 1$ in $\widetilde{O}(T + p)$ operations in $\mathbb{F}_q$ or $\widetilde{O}((T + p) \log q)$ bit operations.*

During the algorithm, we need both to evaluate a sparse polynomial on a geometric progression and to reconstruct a sparse polynomial from these evaluations and its exponents. If $f = \sum_{i=0}^{t-1} c_i x^{e_i} \in \mathbb{F}_q[x]$ is a sparse polynomial, then for any ω

$$\begin{pmatrix} 1 & \cdots & 1 \\ \omega^{e_0} & \cdots & \omega^{e_{t-1}} \\ \omega^{2e_0} & \cdots & \omega^{2e_{t-1}} \\ \vdots & & \vdots \\ \omega^{(t-1)e_0} & \cdots & \omega^{(t-1)e_{t-1}} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_1 \\ \vdots \\ c_{t-1} \end{pmatrix} = \begin{pmatrix} f(1) \\ f(\omega) \\ f(\omega^2) \\ \vdots \\ f(\omega^{t-1}) \end{pmatrix}.$$

This shows that the evaluation is a matrix-vector product and the interpolation the resolution of a linear system, where the matrix is a transposed Vandermonde matrix.

These problems admit algorithms of complexity $\widetilde{O}(t)$ over any finite field through connections to dense polynomial arithmetic in degree t [38, 11] Actually, these algorithms work for more general rings. It is trivial for the matrix-vector product that does not require any inversion in the ring. The resolution of the linear system requires the matrix to be invertible, that is $\omega^{e_i} - \omega^{e_j}$ must be a unit for $i \neq j$. This condition holds when ω is a p-th *principal* root of unity, that is when $\omega^p = 1$ and $\omega^i - 1$ is not a zero divisior for $0 < i < p$. The following fact summarizes these known results.

Fact 3.2. *Let R be a ring, $f = \sum_{i=0}^{t-1} c_i x^{e_i}$ be a sparse polynomial over R, and ω a principal p-th root of unity. Then*

- *evaluating f mod $x^p - 1$ at $1, \omega, \ldots, \omega^{t-1}$, and*

- *retrieving the coefficients of f mod $x^p - 1$ from its set of exponents and $f(1), \ldots, f(\omega^{t-1})$*

can be done in $\widetilde{O}(t \log p)$ operations in R.

We shall use these results over two rings. First, using Fact 3.1 we perform the evaluation on powers of a p-PRU in $\mathbb{F}_q$ to recover the set of exponents modulo p. From these exponents, we rely on Fact 3.2 with a p-PRU $\omega_k \in \mathbb{Z}/q^k\mathbb{Z}$ to recover the polynomial modulo $x^p - 1$ over the larger ring $\mathbb{Z}/q^k\mathbb{Z}$, using this time both evaluation and interpolation. Note that k is carefully chosen so that it allows to recover all the integer coefficients of f mod $(x^p - 1)$. The correctness follows directly from Lemma 2.3 that shows that a p-PRU in $\mathbb{Z}/q^k\mathbb{Z}$ is also principal.

While we completely know f mod $x^p - 1$, some terms of this polynomial come from collisions: That is, two (or more) distinct monomials $c_i x^{e_i}$ and $c_j x^{e_j}$ from f may *collide* modulo p and create the term $(c_i + c_j)x^{e_i \bmod p}$ in f mod $x^p - 1$. We shall overcome this difficulty by a random choice of p that guarantees that with good probability, not too many terms collide. Other terms of f mod $x^p - 1$ are *collision-free*, that is of the form $c_i x^{e_i \bmod p}$. To recover the exponent e_i from these terms, we embed the exponents into its coefficients.

The idea, due to Arnold and Roche [6], is to compute the sparse representations of both f and $f((1 + q^k)x)$, modulo $\langle x^p - 1, q^{2k} \rangle$. Since $(1 + q^k)^{e_i} = 1 + e_i q^k$ mod q^{2k}, a collision-free term $c_i x^{e_i}$ is mapped to $c_i x^{e_i \bmod p}$ in f mod $\langle x^p - 1, q^{2k} \rangle$ and $c_i' x^{e_i \bmod p}$ in $f((1 + q^k)x)$ mod $\langle x^p - 1, q^{2k} \rangle$ where $c_i' = c_i(1 + e_i q^k)$. This allows us to recover both c_i and $e_i = (c_i'/c_i - 1)/q^k$ as soon as k is large enough. More precisely, we need c_i to be a unit and representable in $\mathbb{Z}/q^{2k}\mathbb{Z}$, and $(1 + e_i q^k) \leq q^{2k}$ so that the division by q^k remains over the integers. That is, q must be chosen not to divide any coefficient and $k > \max(\frac{1}{2}\log_q 2H, \log_q D)$.

We note that there is no *a priori* way to distinguish between collision-free terms and colliding terms. For some colliding terms, the recovered value of e_i is clearly wrong since it is not integral or too large, but one cannot avoid recovering unwanted terms in general. This is again taken care of through the choice of p, as in [27, 30], to avoid reconstructing too many erroneous terms.

Fact 3.3. *Given the sparse representation of $f(x)$ mod $\langle x^p - 1, q^{2k} \rangle$ and $f((1+q^k)x)$ mod $\langle x^p - 1, q^{2k} \rangle$ such that q does not divide any coefficient of f mod $x^p - 1$ and $k \geq$*

$\max(\frac{1}{2}\log_q 2H, \log_q D)$, *one can compute a set of tentative terms of* f, *containing all the collision-free terms modulo* $x^p - 1$, *in* $O(T)$ *arithmetic operations.*

INTERPOLATE_MBB given in Algorithm 2 follows the idea from the three previous facts to reach a softly-linear time complexity.

Algorithm 2: INTERPOLATE_MBB

Input : a polynomial $f \in \mathbb{Z}[x]$ represented by an MBB; bounds D, T and H on respectively the degree, the sparsity and the height of f

Output: the sparse representation of $f \in \mathbb{Z}[x]$ with probability $\geq \frac{2}{3}$; otherwise any T-sparse polynomial or FAIL

1   $f^* \leftarrow 0$; $\epsilon \leftarrow 1/(9\lceil\log T\rceil)$

2   $\lambda \leftarrow \max\left(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(T-1)\ln D, \sqrt[5]{\frac{48}{\epsilon}T\ln H}\right)$

    `/* Heuristically `$\frac{2^{58}}{\epsilon^2}$` can be replaced by 1, see`
       `discussion after `Fact 2.2`.`   `*/`

3   **while** $T \geq 1$ **do**

4   Compute a triple (p, q, ω) such that $\omega \in \mathbb{F}_q$ is a p-PRU where p and q are prime numbers and $\lambda < p < 2\lambda$ using Fact 2.2

5   Evaluate $(f - f^*)$ at $1, \omega, \ldots, \omega^{2T-1}$ and compute the exponents of $(f - f^*) \bmod \langle x^p - 1, q\rangle$ using Fact 3.1

6   Compute a p-PRU $\omega_k \in \mathbb{Z}/q^{2k}\mathbb{Z}$ where $k = \lceil\max(\frac{1}{2}\log_q 2H, \log_q D)\rceil$ using Theorem 2.4

7   Evaluate $(f - f^*)$ at $1, \omega_k, \ldots, \omega_k^{T-1}$ and compute the sparse representation of $(f - f^*) \bmod \langle x^p - 1, q^{2k}\rangle$ using Fact 3.2

8   Perform the same step with shifted evaluation points to compute the sparse representation of $(f - f^*)((1 + q^k)x) \bmod \langle x^p - 1, q^{2k}\rangle$

9   Compute tentative terms of $(f - f^*)$ using Fact 3.3

10   Add the tentative terms to f^* ; $T \leftarrow \lfloor T/2\rfloor$

11 **return** f^*

Theorem 3.4. *Algorithm* INTERPOLATE_MBB *works as specified. It requires* $O(T)$ *probes to the MBB,* $\widetilde{O}(T \log DH)$ *operations on integers of size* $O(\log(T \log DH))$, *and* $\widetilde{O}(T \log\log DH)$ *operations on integers of size* $O(\log DH)$. *If the input is an SLP of length* L *and if* H *is also a bound on the absolute values of the constants of the SLP, the bit complexity of the algorithm is* $\widetilde{O}(LT(\log D + \log H))$.

For any $\rho \geq 1$, $O(\rho)$ *repetitions of the algorithm improve the success probability to* $1 - \frac{1}{2^\rho}$.

Correctness. The algorithm has three sources of failure at each iteration. First, the algorithm may fail to produce a triple (p, q, ω) satisfying the conditions. By Fact 2.2, this probability is at most ϵ. Second, the number of collisions of $(f - f^*) \bmod x^p - 1$ may be too large. Fact 2.1 and our choice of λ guarantee that with probability at least $1 - \epsilon$, the number of collisions is at most $\frac{1}{3}t$ where $t \leq T$ is the true sparsity of $(f - f^*)$.

Third, some coefficients of $(f - f^*) \bmod x^p - 1$ may vanish modulo q. Fact 2.2 and our choice of λ guarantee that this probability is at most ϵ. Therefore, each iteration fails with probability at most $3\epsilon = 1/3 \lceil \log T \rceil$, whence the algorithm fails with probability at most $\frac{1}{3}$.

We now prove that, assuming that none of these possible failures happens, $f^* = f$ at the end of the algorithm. Fact 3.1 proves that Line 5 correctly computes the exponents of $(f - f^*) \bmod x^p - 1$. Fact 3.2 proves that Lines 7 and 8 correctly compute the sparse representations of $(f - f^*) \bmod \langle x^p - 1, q^{2k} \rangle$ and its shifted counterpart. Therefore, since k is large enough, Fact 3.3 ensures that Line 9 computes all the collision-free terms of $(f - f^*)$ plus some erroneous terms. By assumption, the number of collisions of $(f - f^*) \bmod x^p - 1$ is at most $\frac{1}{3}t$. Since collisions involve at least two terms, the number of colliding terms in $(f - f^*) \bmod x^p - 1$ is at most $\frac{t}{6}$. Therefore, the tentative terms at Line 9 contain at least $\frac{2}{3}t$ correct terms and at most $\frac{1}{6}t$ incorrect terms. In other words, the number of terms in $(f - f^*)$ at the end of the iteration is at most $t - \frac{2}{3}t + \frac{1}{6}t = \frac{1}{2}t$. After $\log T$ iterations, $f = f^*$.

To improve the success probability, we repeat the algorithm $48\rho/\log e$ times and return the majority polynomial. Let C be the number of repetitions that produce the correct polynomial. Since each repetition is correct with probability at least $\frac{2}{3}$, $\mathbb{E}[C] = \frac{32\rho}{\log e}$. Therefore, by Chernoff bound, the probability that the correct polynomial is produced by less than half of the repetitions is $\Pr[C \le \frac{24\rho}{\log e}] = \Pr[C \le (1 - \frac{1}{4})\mathbb{E}[C]] \le \exp(-(\frac{1}{4})^2 \mathbb{E}[C]/2) = \frac{1}{2^\rho}$. □

Complexity. Each iteration require $3T$ probes to the MBB (with the current value of T). Hence the total number of probes is $< 6T$. The evaluations of f^* at powers of ω and ω_k require $\widetilde{O}(t \log p) = \widetilde{O}(T \log \log DH)$ operations in $\mathbb{F}_q$ or $\mathbb{Z}/q^{2k}\mathbb{Z}$ by Fact 3.2. Apart from the evaluations, Line 5 requires $\widetilde{O}(p) = \widetilde{O}(T \log DH)$ operations in $\mathbb{F}_q$ using Fact 3.1 and Lines 7 and 8 require $\widetilde{O}(T \log p) = \widetilde{O}(T \log \log DH)$ operations in $\mathbb{Z}/q^{2k}\mathbb{Z}$ using Fact 3.2.

The bit cost of each arithmetic operation is $\widetilde{O}(\log q) = \widetilde{O}(\log(T \log D) + \log \log H))$ for those in $\mathbb{F}_q$, and $\widetilde{O}(k \log q) = \widetilde{O}(\log D + \log H)$ for those in $\mathbb{Z}/q^{2k}\mathbb{Z}$. If the MBB is implemented with an SLP, the overall bit complexity, dominated by the evaluations of the SLP, is $\widetilde{O}(LT(\log D + \log H))$. Note that computing p, q, ω and ω_k is cheap, since p, q are rather small. □

Our algorithm is randomized of Monte Carlo type since it may return an incorrect answer, in addition to fail. To get a Las Vegas variant, the algorithm should only be allowed to fail. For, we need a verification procedure that itself is a Las Vegas algorithm. The problem to solve is then: Given an MBB for a polynomial f and a sparse polynomial f^*, determine whether $f = f^*$. Bläser et al. [8] provide deterministic algorithms for this task but with polynomial, and not quasi-linear complexity. Another approach relies on the same tools as Ben-Or–Tiwari algorithm. If both f and f^* have sparsity at most T and degree at most D, and ω is an element of order at least D, then $f - f^*$ vanishes on $1, \omega, \ldots, \omega^{2T-1}$ if and only if $f = f^*$ (*cf.* for instance [3]). It is deterministic as long as an element of large order can be computed deterministically.

For a polynomial over $\mathbb{Z}$, we must evaluate f and f^* modulo some integer m to avoid expression swell. As before, we can produce a triple (p, q, ω) such that ω is a p-PRU in $\mathbb{F}_q$. Since ω should have order $\geq D$, we take a random prime $p \geq D$, and $q \geq H$ so that the coefficients do not vanish modulo q. This can be done in time $\mathsf{polylog}(D + H)$. Then, evaluating f^* on $1, \omega, \ldots, \omega^{2T-1}$ requires $2T$ probes to the MBB for f, and $O(T \log D)$ operations in $\mathbb{F}_q$ for f^*. If f is represented by an SLP of length L, the bit complexity becomes $\widetilde{O}(LT \log(D + H) + T \log(D) \log(D + H))$. Note that this complexity is quadratic in $\log D$.

Altogether, we obtain a Las Vegas algorithm using $O(T)$ probes, $O(T \log D)$ operations in $\mathbb{F}_q$ and $\mathsf{polylog}(D + H)$ bit operations, with a constant probability of failure. If f is represented by an SLP, the bit complexity is $\widetilde{O}(LT \log(D + H) + T \log(D) \log(D + H))$. Using repetition, we obtain an algorithm that never fails, with the same *expected* complexity.

It is an intriguing open question whether a quasi-linear Las Vegas algorithm exists. In particular, can we verify an equality $f = f^*$ where f is given by an SLP and f^* is sparse, in quasi-linear time?

4   Exact division

Given two sparse polynomials f and g such that g divides f, the problem of computing f/g can be seen as a sparse interpolation of a specific SLP that has a single division. As shown in Giorgi et al. [19] some sparse interpolation algorithms can be carefully adapted to produce division algorithms if there is no remainder. As the interpolation algorithms they rely on, these division algorithms are not quasi-linear in the input plus the output bit-size. In this section we show how to adapt of our quasi-linear interpolation algorithm to derive fast sparse polynomial exact division. As a result, we obtain the first quasi-linear exact division algorithm for sparse polynomial over the integers.

There are three main difficulties in adapting our interpolation algorithm. First, no bound is given for $\#(f/g)$ except the potentially exponential degree one. Second, we do not know the height of f/g while the interpolation algorithm depends on it. Last, to evaluate the quotient f/g at a root of unity ω, we compute both $f(\omega)$ and $g(\omega)$ and perform the division. Hence, ω must not be a root of g.

To overcome the first difficulty, we use the same method as Giorgi et al. [18, 19]. We guess a sparsity bound for the quotient, interpolate a candidate quotient assuming the bound, and check its correctness *a posteriori* with a probabilistic verification. In case of failure we double the sparsity bound and start again.

Besides verifying products of sparse polynomials, we will also need in our algorithm an efficient verification of sparse polynomial product modulo a binomial. Such algorithms have been recently proposed by some of the authors in [20], and we recall the useful results below.

Fact 4.1 (Giorgi et al. [20]). *There exists a Monte Carlo algorithm that, given three t-sparse degree-D polynomials $f, g, h \in \mathbb{Z}[x]$ of height $\leq H$, and $\rho \geq 1$, verify if $f = gh$. The algorithm can give a wrong answer with probability at most $\frac{1}{2^\rho}$ when $f \neq gh$. Its bit*

complexity is $\widetilde{O}\big(t(\log D + \log H + \rho) + \rho^4\big)$.

There exists a Monte Carlo algorithm that similarly tests if $f = gh$ mod $x^D - 1$, *with the same error probability and bit complexity* $\widetilde{O}\big(t\rho \log D + t \log H + \rho^4 \log^3 D\big)$.

A similar *guess and check* method can be used to determine an appropriate bound for the height of the quotient: Start with a small bound and increase it when necessary. Indeed, Line 7 of algorithm INTERPOLATE_MBB correctly computes the polynomial modulo $x^p - 1$ as soon as q^{2k} is greater than its height. There, verifying the sparse product modulo $x^p - 1$ allows us to determine if the bound on the height is large enough. This method is necessary as the bound we have for the height is exponential.

Fact 4.2 (Giorgi et al. [19]). *Let* $f, g, q \in \mathbb{Z}[x]$ *be three sparse polynomials such that* $f = gq$. *Then the height* H_q *of* q *satisfies* $H_q \leq (H_g + 1)^{\lceil \frac{t-1}{2} \rceil} H_f$ *where* $t = \#q$ *and* H_f, H_g *are the respective heights of* f *and* g.

For the last difficulty, we want $g(\omega) \neq 0$ for any pth primitive root of unity ω in $\mathbb{F}_q$. That is, we want g to be coprime with the pth cyclotomic polynomial $\Phi_p = \sum_{i=0}^{p-1} x^i$ in $\mathbb{F}_q[x]$. In $\mathbb{Z}[x]$, if p is a prime larger than $\#g$ such that g mod $x^p - 1 \neq 0$, then g and Φ_p are coprime. If p is taken at random and large enough, namely $p = \Omega(\#g \log(\deg g))$, Fact 2.1 ensures that g mod $x^p - 1 \neq 0$ with good probability. Then, g and Φ_p are coprime in $\mathbb{F}_q[x]$ if and only if q does not divide their resultant, an integer bounded by $(\#g \cdot H_g)^{p-1}$ where H_g is the height of g. We can therefore choose two primes p and q so that g and Φ_p are coprime in $\mathbb{F}_q[x]$ with good probability, using Fact 2.2.

We first describe an algorithm to compute an exact quotient with a given bound on its sparsity but no precise bound on its height.

The algorithm can return an erroneous polynomial by adding false terms. However this polynomial cannot be much larger than the correct polynomial.

Lemma 4.3. *Algorithm* BOUNDED_SPARSITY_DIVISION *always returns a polynomial with at most* $2T$ *terms and height at most* $T \cdot tH$ *where* t *and* H *are the actual sparsity and height of the quotient we intend to compute.*

Proof. For the sparsity, Line 6 uses a Vandermonde system to interpolate a sparse polynomial of sparsity at most T and cannot compute more than T monomials. Therefore, as T is divided by 2 every time we add new terms to h, the result has at most $2T$ terms.

For the height, only erroneous terms can have coefficients larger than H. However those terms necessarily come from collisions. Hence at each iteration, the sum of the erroneous terms is at most equal to the sum of the terms of $f/g - h$. Initially, $h = 0$ and the sum is bounded by tH. At each iteration, erroneous terms can at most double the sum. After $\lceil \log T \rceil$ iteration, the sum is bounded by $T \cdot tH$ and so is the height of h. $\square$

Theorem 4.4. *Algorithm* BOUNDED_SPARSITY_DIVISION *works as specified. Its bit complexity is* $\widetilde{O}((T + \#f + \#g)(\log D + \log H))$ *where* $D = \deg(f)$ *and* H *bounds the height of* f, g *and* f/g.

For any $\rho \geq 1$, $O(\rho)$ *repetitions of the algorithm improve the success probability to* $1 - \frac{1}{2^\rho}$.

Algorithm 3: BOUNDED_SPARSITY_DIVISION

Input : two sparse polynomials $f, g \in \mathbb{Z}[x]$ such that f has degree D and g
divides f; an integer T

Output: f/g with probability at least $\frac{2}{3}$, if $T \geq \#(f/g)$

---

1   $H_{max} \leftarrow (1 + H_g)^{\lceil \frac{1}{2}(T-1)\rceil} \cdot H_f$ where H_f, H_g are the heights of f and g

2   $\epsilon \leftarrow \frac{1}{15}(\lceil \log T \rceil + \lceil \log \log H_{max} \rceil); C \leftarrow H_{max} \cdot \#gH_g$

3   $\lambda \leftarrow \max\left(\frac{2^{58}}{\epsilon^2}, \frac{5}{\epsilon}(\max(T, \#g) - 1)\ln D, \sqrt[4]{\frac{96}{\epsilon}}\ln C \right)$

4   $h \leftarrow 0; H_0 \leftarrow H_g + 1$

5   **while** $T \geq 1$ **do**

6   Compute $h_p = (f/g - h) \bmod \langle x^p - 1, q^{2k} \rangle$ as in INTERPOLATE_MBB, where
      $\lambda < p < 2\lambda$, $q \leq \lambda^6$ and $k = \lceil \max(\frac{1}{2}\log_q(2H_0H_f), \log_q D)\rceil$

7   Test if $f \bmod x^p - 1 = g \times (h_p + h) \bmod x^p - 1$, with error probability $\leq \frac{1}{\epsilon}$,
      using Fact 4.1

8   **if** *the test returns* TRUE **then**

9   Compute tentative terms of $f/g - h$

10   Add the terms of height $\leq H_{max}$ to h

11        $T \leftarrow \lfloor T/2 \rfloor$

12   **else** $H_0 \leftarrow H_0^2$

13 **return** r

Correctness. The algorithm may fail for five distinct reasons. The first three reasons are the same as in INTERPOLATE_MBB: It may fail to compute the triple (p, q, ω) required to compute h_p; The prime p may cause too many collisions in $f/g - h \bmod (x^p - 1)$; some terms of $f/g - h \bmod (x^p - 1)$ may vanish modulo q. The two other sources of failure are specific to this algorithm: One of the powers of ω or ω_k may be a root of g; The test at Line 7 may fail to detect an error.

The choice of $\lambda \geq \sqrt[4]{\frac{96}{\epsilon}}\ln C$ implies $\lambda \geq \sqrt[5]{\frac{48}{\epsilon}}\ln(C^{2\lambda})$. Facts 2.1 and 2.2 ensure that, with probability at least $1 - 3\epsilon$, the algorithm successfully produces a triple (p, q, ω) such that p does not cause too many collisions and q does not divide an unknown integer of value at most C^p. If p does not cause too many collisions, $g \bmod x^p - 1 \neq 0$. Since $\#g < p$, g and $\Phi_p = \sum_{i=0}^{p-1} x^i$ are coprime in $\mathbb{Z}[x]$. The resultant of g and Φ_p is at most $(\#gH_g)^p$. Moreover, since H_{max} bounds the height of both h and f/g using Fact 4.2, and since $p > T$, the height of $(f/g - h) \bmod x^p - 1$ is at most H_{max}^p. Hence with probability at least $1 - \epsilon$, q does not divide the resultant of g and Φ_p nor any coefficient of $(f/g - h) \bmod x^p - 1$. In particular, g and Φ_p remain coprime in $\mathbb{F}_q$ and so in $\mathbb{Z}/q^{2k}\mathbb{Z}$ since p-PRU in $\mathbb{Z}/q^{2k}$ are also p-PRU in $\mathbb{F}_q$.

Altogether, the four following properties hold with probability at least $1 - 4\epsilon$: The algorithm succeeds in producing two primes p, q and $\omega \in \mathbb{F}_q$; g and Φ_p are coprime in $\mathbb{F}_q[x]$ and in $\mathbb{Z}/q^{2k}\mathbb{Z}$; There are few collisions in $f/g - h$ modulo $x^p - 1$; q does not divide any of the coefficients of $(f/g - h) \bmod x^p - 1$.

If all these conditions hold, we can use Facts 3.1 and 3.2 to compute h_p. The choice

of k implies that q^{2k} is larger than twice the height of $f/g - h$ as soon as H_0 is larger than the (unknown) height H of f/g. In that case, the equality $h_p = f/g - h$ holds in $\mathbb{Z}[x]$ and the test at Line 7 returns TRUE. Computing tentative terms and updating h can then be done exactly as in INTERPOLATE_MBB.

If $H_0 < H$, there are two possibilities. Either $h_p \neq f/g - h \bmod x^p - 1$ in $\mathbb{Z}[x]$. With probability at least $1 - \epsilon$, the test detects that and H_0 is squared. Or the equality indeed holds. This means that the terms of $f/g - h$ that have a larger height collide modulo $x^p - 1$. Hence, the collision-free terms are correctly computed.

Consequently, the loop works correctly with probability $1 - 5\epsilon$: Either the number of terms that remain to be computed is halved, or the height bound is squared if it was too small. At most $\lceil \log \log H \rceil \le \lceil \log \log H_{max} \rceil$ iterations where the test returns FALSE are needed to get to a correct bound $H_0 \ge H$, and at most $\lceil \log T \rceil$ iterations where the test returns TRUE are needed to to compute all the coefficients. Therefore the algorithm performs at most $(\lceil \log T \rceil + \lceil \log \log H_{max} \rceil)$ iterations. Its success probability is at least $1 - 5\epsilon(\lceil \log T \rceil + \lceil \log \log H_{max} \rceil) \ge \frac{2}{3}$. To improve the success probability, we repeat the algorithm $48\rho/\log e$ times and return the majority polynomial, as in INTERPOLATE_MBB. $\qquad\square$

Complexity. Since the number of iterations is logarithmic in the input and output size, the complexity of the algorithm is given by the complexity of one iteration. As in INTERPOLATE_MBB, the algorithm requires $\widetilde{O}(T + p)$ operations in $\mathbb{F}_q$ and $\widetilde{O}(T \log p)$ operations $\mathbb{Z}/q^{2k}\mathbb{Z}$ for the evaluations of h, computing the exponents modulo p and then retrieving the coefficients and the entire exponents. The evaluations of f/g require $\widetilde{O}((T + \#f + \#g) \log p)$ operations in both domains by Fact 3.2 plus $O(\#f + \#g)$ operations in $\mathbb{Z}$ to reduce the initial coefficients and degree. As the height of an erroneous answer is at most $T^2 H$ by Lemma 4.3, the maximal value of q^{2k} is $O(T^2 H + D)$. Therefore arithmetic operations in $\mathbb{Z}/q^{2k}\mathbb{Z}$ have bit cost $\widetilde{O}(\log H + \log D)$. Moreover the choice of λ ensures that $p = \widetilde{O}((T + \#g)(\log D + \log H))$. As q is polynomial in p this leads to a total bit complexity of $\widetilde{O}((T + \#f + \#g)(\log D + \log H))$. $\qquad\square$

Our main division algorithm uses BOUNDED_SPARSITY_DIVISION with growing sparsity bound until a result is found.

Algorithm 4: EXACT_DIVISION

Input : $f, g \in \mathbb{Z}[x]$, such that g divides f, $\rho \ge 1$
Output: f/g with probability at least $1 - \frac{1}{2^{\rho+1}}$

1   $T \leftarrow 1$
2   **while** TRUE **do**
3      $T \leftarrow 2T$
4   Compute $O(\rho)$ candidates h for f/g using Algorithm 3 with sparsity bound T and keep the most frequent one
5   Test if $f = gh$ using the algorithm from Fact 4.1, setting its failure probability to $\frac{1}{2^{\rho+1}T}$
6   If the test returns TRUE, **return** h

Theorem 4.5. *Let f, g be sparse polynomials in $\mathbb{Z}[x]$ such that g divides f, H be a bound on the height of f, g and f/g, and $\rho \geq 1$. With probability at least $1 - \frac{1}{2^\rho}$, Algorithm EXACT_DIVISION returns f/g in $\widetilde{O}\big((\#(f/g) + \#f + \#g)(\log D + \log H + \rho) + \rho^4\big)$ bit operations.*

Proof. The probability $1 - \frac{1}{2^\rho}$ concerns both the correctness and the complexity of the algorithm. We prove that each of them holds independently with probability $\geq 1 - \frac{1}{2^{\rho+1}}$.

The algorithm is incorrect when $f \neq gh$. This happens if at some iteration, the candidate quotient is incorrect but the verification algorithm fails to detect it. Since each verification fails with probability at most $\frac{1}{2^{\rho+1}T}$ and values of T range over powers of two, the algorithm is correct with probability at least $1 - \frac{1}{2^{\rho+1}}$.

For the complexity we first need to bound the number of iterations. Since the values of T are powers of two, the first value $\geq \#(f/g)$ is at most $2\#(f/g)$. As soon as T reaches this value, the return value is actually f/g with probability at least $1 - \frac{1}{2^{\rho+1}}$ according to Theorem 4.4 when the number of candidates is $\geq 48(\rho+1)/\log e$. In that case, the test which is only one-sided error, succeeds and the algorithm returns $h = f/g$. That is, with probability at least $1 - \frac{1}{2^{\rho+1}}$, the number of iterations is $O(\log \#(f/g))$. Even with false sparsity, Lemma 4.3 ensures that the size of the candidate quotients is at most quasi-linear in the size of the actual quotient. Therefore we can apply Theorem 4.4 to obtain the claimed complexity with probability at least $1 - \frac{1}{2^{\rho+1}}$. $\qquad\square$

Acknowledgements

References

[1] Amir Akbary and Kyle Hambrook. 2015. A variant of the Bombieri-Vinogradov theorem with explicit constants and applications. *Math. Comp.* 84, 294 (2015), 1901–1932. DOI: 10.1090/S0025-5718-2014-02919-0. Referenced on page 7.

[2] Noga Alon and Yishay Mansour. 1995. epsilon-discrepancy sets and their application for interpolation of sparse polynomials. *Inform. Process. Lett.* 54, 6 (1995), 337–342. DOI: 10.1016/0020-0190(95)00032-8. Referenced on page 2.

[3] Andrew Arnold. 2016. *Sparse Polynomial Interpolation and Testing*. Ph. D. Dissertation. University of Waterloo. URL: http://hdl.handle.net/10012/10307. Referenced on pages 3, 7 and 12.

[4] Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. 2014. Sparse interpolation over finite fields via low-order roots of unity. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)*. Association for Computing Machinery, 27–34. DOI: 10.1145/2608628.2608671. Referenced on pages 3 and 7.

[5] Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. 2015. Faster sparse multivariate polynomial interpolation of straight-line programs. *Journal of Symbolic Computation* (2015). DOI: 10.1016/j.jsc.2015.11.005. Referenced on page 3.

[6] Andrew Arnold and Daniel S. Roche. 2015. Output-Sensitive Algorithms for Sumset and Sparse Polynomial Multiplication. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation* (Bath, United Kingdom) *(ISSAC '15)*. ACM, 29–36. DOI: 10.1145/2755996.2756653. Referenced on pages 3, 5, 7 and 10.

[7] Michael Ben-Or and Prasoon Tiwari. 1988. A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, 301–309. DOI: 10.1145/62212.62241. Referenced on pages 2 and 5.

[8] Markus Bläser, Moritz Hardt, Richard J. Lipton, and Nisheeth K. Vishnoi. 2009. Deterministically Testing Sparse Polynomial Identities of Unbounded Degree. *Inform. Process. Lett.* 109, 3 (2009), 187–192. DOI: 10.1016/j.ipl.2008.09.029. Referenced on page 12.

[9] Leo I. Bluestein. 1970. A Linear Filtering Approach to the Computation of Discrete Fourier Transform. *IEEE Transactions on Audio and Electroacoustics* 18, 4 (1970), 451–455. DOI: 10.1109/TAU.1970.1162132. Referenced on page 9.

[10] Markus Bläser and Gorav Jindal. 2014. A new deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation.* Association for Computing Machinery, New York, NY, USA. DOI: 10.1145/2608628.2608648. Referenced on page 2.

[11] Alin Bostan, Grégoire Lecerf, and Éric Schost. 2003. Tellegen's Principle into Practice. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation* (Philadelphia, PA, USA) *(ISSAC '03)*. ACM, 37–44. DOI: 10.1145/860854.860870. Referenced on page 10.

[12] Annie Cuyt and Wen-shin Lee. 2011. Sparse interpolation of multivariate rational functions. *Theoretical Computer Science* 412, 16 (2011), 1445–1456. DOI: 10.1016/j.tcs.2010.11.050. Referenced on page 2.

[13] Jean-Louis Dornstetter. 1987. On the Equivalence Between Berlekamp's and Euclid's Algorithms. *IEEE Transactions on Information Theory* 33, 3 (1987), 428–431. DOI: 10.1109/TIT.1987.1057299. Referenced on page 9.

[14] Sanchit Garg and Éric Schost. 2009. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science* 410, 27-29 (2009), 2659–2662. DOI: 10.1016/j.tcs.2009.03.030. Referenced on pages 2 and 5.

[15] Mickaël Gastineau and Jacques Laskar. 2015. Parallel sparse multivariate polynomial division. In *Proceedings of the 2015 International Workshop on Parallel Symbolic Computation (PASCO '15)*. Association for Computing Machinery, New York, NY, USA, 25–33. DOI: 10.1145/2790282.2790285. Referenced on page 3.

[16] Mark Giesbrecht and Daniel S. Roche. 2011. Diversification improves interpolation. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation - ISSAC '11*. ACM Press, San Jose, California, USA, 123. DOI: 10.1145/1993886.1993909. Referenced on page 2.

[17] Pascal Giorgi, Bruno Grenet, Armelle Perret du Cray, and Daniel S. Roche. 2022. Random primes in arithmetic progressions. arXiv: 2202.05955. Referenced on page 7.

[18] Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. 2020. Essentially optimal sparse polynomial multiplication. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation* (Kalamata, Greece) *(ISSAC'20)*. 202–209. DOI: 10.1145/3373207.3404026. Referenced on pages 4, 6 and 13.

[19] Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. 2021. On exact division and divisibility testing for sparse polynomials. In *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation (ISSAC'21)*. 163–170. DOI: 10.1145/3452143.3465539. Referenced on pages 4, 13 and 14.

[20] Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. 2022. Polynomial modular product verification and its implications. *Journal of Symbolic Computation* (2022), to appear. Referenced on pages 4, 6 and 13.

[21] Dima Yu Grigoriev, Marek Karpinski, and Michael F. Singer. 1990. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.* 19, 6 (1990), 1059–1063. DOI: 10.1137/0219073. Referenced on page 2.

[22] Joris van der Hoeven. 2020. Probably faster multiplication of sparse polynomials. (2020). HAL: hal-02473830. Referenced on page 3.

[23] Joris van der Hoeven and Grégoire Lecerf. 2015. Sparse Polynomial Interpolation in Practice. *ACM Communications in Computer Algebra* 48, 3/4 (2015), 187–191. DOI: 10.1145/2733693.2733721. Referenced on pages 2, 3 and 5.

[24] Joris van der Hoeven and Grégoire Lecerf. 2019. Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields. (2019). HAL: hal-02382117. Referenced on page 3.

[25] Joris van der Hoeven and Grégoire Lecerf. 2021. On sparse interpolation of rational functions and gcds. *ACM Communications in Computer Algebra* 55, 1 (2021), 1–12. DOI: 10.1145/3466895.3466896. Referenced on page 2.

[26] Ming-Deh A. Huang and Ashwin J. Rao. 1999. Interpolation of Sparse Multivariate Polynomials over Large Finite Fields with Applications. *Journal of Algorithms* 33, 2 (1999), 204–228. DOI: 10.1006/jagm.1999.1045. Referenced on page 2.

[27] Qiao-Long Huang. 2019. Sparse Polynomial Interpolation over Fields with Large or Zero Characteristic. In *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation (ISSAC '19)*. ACM Press, Beijing, China, 219–226. DOI: 10.1145/3326229.3326250. Referenced on pages 3, 5 and 10.

[28] Qiao-Long Huang. 2020. Sparse Polynomial Interpolation Based on Derivative. (2020). arXiv: 2002.03708. Referenced on pages 3 and 5.

[29] Qiao-Long Huang. 2021. Sparse polynomial interpolation based on diversification. *Science China Mathematics* (2021). DOI: 10.1007/s11425-020-1791-5. Referenced on pages 2 and 5.

[30] Qiao-Long Huang and Xiao-Shan Gao. 2019. Revisit Sparse Polynomial Interpolation Based on Randomized Kronecker Substitution. In *Computer Algebra in Scientific Computing*. Springer International Publishing, 215–235. DOI: 10.1007/978-3-030-26831-2_15. Referenced on pages 2 and 10.

[31] Qiao-Long Huang and Xiao-Shan Gao. 2020. Faster interpolation algorithms for sparse multivariate polynomials given by straight-line programs. *Journal of Symbolic Computation* 101 (2020), 367–386. DOI: 10.1016/j.jsc.2019.10.005. Referenced on pages 3, 5 and 7.

[32] Seyed Mohammad Mahdi Javadi and Michael Monagan. 2010. Parallel sparse polynomial interpolation over finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation (PASCO '10)*. Association for Computing Machinery, New York, NY, USA, 160–168. DOI: 10.1145/1837210.1837233. Referenced on page 2.

[33] Stephen C. Johnson. 1974. Sparse polynomial arithmetic. *SIGSAM Bulletin* 8, 3 (1974), 63–71. DOI: 10.1145/1086837.1086847. Referenced on page 3.

[34] Erich L. Kaltofen and Wen-shin Lee. 2003. Early termination in sparse interpolation algorithms. *Journal of Symbolic Computation* 36, 3-4 (2003), 365–400. DOI: 10.1016/S0747-7171(03)00088-9. Referenced on page 2.

[35] Erich L. Kaltofen. 2010. Fifteen years after DSC and WLSS2: What parallel computations I do today [invited lecture at PASCO 2010]. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation* (Grenoble, France) *(PASCO '10)*. ACM, 10–17. DOI: 10.1145/1837210.1837213. Referenced on page 4.

[36] Erich L. Kaltofen, Yagati N. Lakshman, and John-Michael Wiley. 1990. Modular rational sparse multivariate polynomial interpolation. In *Proceedings of the international symposium on Symbolic and algebraic computation (ISSAC '90)*. ACM Press, Tokyo, Japan, 135–139. DOI: 10.1145/96877.96912. Referenced on page 3.

[37] Erich L. Kaltofen and Michael Nehring. 2011. Supersparse black box rational function interpolation. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation.* Association for Computing Machinery, New York, NY, USA, 177–186. DOI: 10.1145/1993886.1993916. Referenced on page 2.

[38] Erich L. Kaltofen and Lakshman Yagati. 1988. Improved Sparse Multivariate Polynomial Interpolation Algorithms. In *Symbolic and Algebraic Computation.* Springer Berlin Heidelberg, 467–474. DOI: 10.1007/3-540-51084-2_44. Referenced on pages 2 and 10.

[39] Erich L. Kaltofen and Zhengfeng Yang. 2007. On exact and approximate interpolation of sparse rational functions. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation (ISSAC '07).* ACM Press, Waterloo, Ontario, Canada, 203. DOI: 10.1145/1277548.1277577. Referenced on page 2.

[40] Leopold Kronecker. 1882. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *Journal für die reine und angewandte Mathematik* 92 (1882), 1–122. Referenced on page 4.

[41] Yishay Mansour. 1995. Randomized Interpolation and Approximation of Sparse Polynomials. *SIAM J. Comput.* 24, 2 (1995), 357–368. DOI: 10.1137/S0097539792239291. Referenced on page 2.

[42] Michael Monagan and Roman Pearce. 2007. Polynomial Division Using Dynamic Arrays, Heaps, and Packed Exponent Vectors. In *Computer Algebra in Scientific Computing (CASC '07).* 295–315. DOI: 10.1007/978-3-540-75187-8_23. Referenced on page 3.

[43] Michael Monagan and Roman Pearce. 2009. Parallel sparse polynomial multiplication using heaps. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation (ISSAC'09).* 263–270. DOI: 10.1145/1576702.1576739. Referenced on page 3.

[44] Michael Monagan and Roman Pearce. 2011. Sparse polynomial division using a heap. *Journal of Symbolic Computation* 46, 7 (2011). DOI: 10.1016/j.jsc.2010.08.014. Referenced on page 3.

[45] Hirokazu Murao and Tetsuro Fujise. 1996. Modular Algorithm for Sparse Multivariate Polynomial Interpolationand its Parallel Implementation. *Journal of Symbolic Computation* 21, 4-6 (1996), 377–396. DOI: 10.1006/jsco.1996.0020. Referenced on page 2.

[46] Vasileios Nakos. 2020. Nearly Optimal Sparse Polynomial Multiplication. *IEEE Transactions on Information Theory* 66, 11 (2020), 7231–7236. DOI: 10.1109/TIT.2020.2989385. Referenced on page 3.

[47] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology – EUROCRYPT '99 (Lecture Notes in Computer Science),* Jacques Stern (Ed.). Springer, Berlin, Heidelberg, 223–238. DOI: 10.1007/3-540-48910-X_16. Referenced on page 5.

[48] Daniel S. Roche. 2018. What Can (and Can't) we Do with Sparse Polynomials?. In *Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation (ISSAC'18)*. ACM, 25–30. DOI: 10.1145/3208976.3209027. Referenced on page 3.

[49] Bruno Rousselet. 1985. Estimations du type Brun-Titchmarsh. *Groupe d'étude en théorie analytique des nombres* 1, 37 (1985), 1. Referenced on page 6.

[50] Arnold Schönhage. 1971. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica* 1 (06 1971), 139–144. DOI: 10.1007/BF00289520. Referenced on page 9.

[51] Alisa Sedunova. 2018. A partial Bombieri–Vinogradov theorem with explicit constants. *Publications mathématiques de Besançon. Algèbre et théorie des nombres* (2018), 101–110. DOI: 10.5802/pmb.24. Referenced on page 7.

[52] Richard Zippel. 1990. Interpolating polynomials from their values. *Journal of Symbolic Computation* 9, 3 (1990), 375–403. DOI: 10.1016/S0747-7171(08)80018-1. Referenced on page 2.