On Exact Division and Divisibility Testing for Sparse Polynomials

Pascal Giorgi Bruno Grenet Armelle Perret du Cray
LIRMM, Univ. Montpellier, CNRS
Montpellier, France
{pascal.giorgi,bruno.grenet,armelle.perret-du-cray}@lirmm.fr

May 19, 2021

Abstract

No polynomial-time algorithm is known to test whether a sparse polynomial G divides another sparse polynomial F. While computing the quotient $Q = F \operatorname{quo} G$ can be done in polynomial time with respect to the sparsities of F, G and Q, this is not yet sufficient to get a polynomial-time divisibility test in general. Indeed, the sparsity of the quotient Q can be exponentially larger than the ones of F and G. In the favorable case where the sparsity $\#Q$ of the quotient is polynomial, the best known algorithm to compute Q has a non-linear factor $\#G\#Q$ in the complexity, which is not optimal.

In this work, we are interested in the two aspects of this problem. First, we propose a new randomized algorithm that computes the quotient of two sparse polynomials when the division is exact. Its complexity is quasi-linear in the sparsities of F, G and Q. Our approach relies on sparse interpolation and it works over any finite field or the ring of integers. Then, as a step toward faster divisibility testing, we provide a new polynomial-time algorithm when the divisor has a specific shape. More precisely, we reduce the problem to finding a polynomial S such that QS is sparse and testing divisibility by S can be done in polynomial time. We identify some structure patterns in the divisor G for which we can efficiently compute such a polynomial S.

1 Introduction

The existence of quasi-optimal algorithms for most operations on dense polynomials yields a strong base for fast algorithms in computer algebra [10] and more generally in computational mathematics. The situation is different for algorithms involving sparse polynomials. Indeed, the sparse representation of a polynomial $F = \sum_{i=0}^{D} f_i X^i \in R[X]$ is a list of pairs (e_i, f_{e_i}) such that each f_{e_i} is nonzero. Therefore, the size of the sparse representation of F is $O(\#F(B + \log D))$ bits, where B and $\#F$ bound respectively the size of the coefficients and the number of nonzero coefficients of F. Polynomial-time algorithms for sparse polynomials need to have a (poly-)logarithmic dependency on the degree. On the one hand, several NP-hardness results rule out the existence of such fast algorithms unless P = NP, for instance for GCD computations [27]. On the other hand, polynomial-time algorithms are known for many important operations such as multiplication, division or sparse interpolation. We refer to Roche's survey [30] for a thorough discussion on their complexity and on the remaining major open problems.

The main difficulty with sparse polynomial operations is the fact that the size of the output does not exclusively depend on the size of the inputs, contrary to the dense case. For instance, the product of two polynomials F and G has at most $\#F\#G$ nonzero coefficients. But it may have as few as 2 nonzero coefficients [12]. The size growth can be even more dramatic for sparse polynomial division. For instance, the quotient of $F = X^D - 1$ by $G = X - 1$ is $F/G = \sum_{i=1}^{D-1} X^i$. The output can therefore be exponentially larger than the inputs. Such a growth is a major difficulty to design efficient algorithms for Euclidean division since it is hard to predict the sparsity of the quotient and the remainder, which can range from constant to exponential.

One important line of work with sparse polynomials is to find algorithms with a quasi-optimal bit complexity $\tilde{O}(T(\log D + \log C))$ where T is the number of nonzero coefficients of the input and output, D the degree and $\log C$ a bound on the coefficient bitsize. The problem is trivial for addition and subtraction. For multiplication, though many algorithms have been proposed [3, 29, 23, 16, 15, 21, 8, 26], none of them was quasi-optimal in the general case. Only recently, we proposed a quasi-optimal algorithm for the multiplication of sparse polynomials over finite fields of large characteristic or over the integers [12]. We note that similar results have been given in a more recent preprint, assuming some heuristics [14].

These fast output-sensitive multiplication algorithms strongly rely on *sparse polynomial interpolation*. In the latter problem, a sparse polynomial is implicitly represented by either a *straight-line program* (SLP) or a *blackbox*. Though efficient output-sensitive algorithms exist in the blackbox model [22] they are not well suited for sparse polynomial arithmetic since one probe of the blackbox is assumed to take a constant time while it is not in our case. Using sparse interpolation algorithms on SLP is not a trivial solution either since no quasi-optimal bit complexity bound is known despite the remarkable recent progress [19, 1, 5, 4, 2, 17, 11, 9, 6, 18]. The best known result due to Huang [19] has bit complexity $\tilde{O}(L(T \log D \log C))$ to interpolate an SLP of length L representing a T-sparse polynomial of degree at most D with coefficient of size $\log C$.

In this work, we are interested to use fast sparse interpolation to derive a better complexity bound for sparse polynomial division, in the special case where the division is exact. As a second goal, we make progress on the very related problem of testing the divisibility of two sparse polynomials.

1.1 Previous work

Euclidean division of sparse polynomials. Let $F = GQ+R \in \mathbb{K}[X]$ where F and G are two polynomials with at most T nonzero coefficients ($\#F$, $\#G \leq T$), $D = \deg(F) > n = \deg G$, and $\deg R < \deg G$. Computing Q and R through classic Euclidean division requires $O(\#G\#Q)$ operations in $\mathbb{K}$. Yet keeping track of the coefficients of the remainder during the computation dominates the cost, due to many exponent comparisons. The total complexity is $O(\#F + \#Q(\#G)^2)$ using sorted lists, or $O(\#F + \#Q\#G \log(\#F + \#Q\#G))$ using binary heaps or the geobucket structure [31]. Heap technique has been improved to further lower down the size of the heap. Johnson proposes an algorithm that uses a heap of size $\#Q+1$ [21], and Monagan and Pearce provide a variant that maintains a heap of size $O(\#G)$ [25]. The best solution to date for sparse polynomial division is to switch from a quotient heap to a divisor heap whenever the quotient is getting larger than the divisor. The complexity becomes $O(\#F + \#Q\#G \log \min(\#Q, \#G))$ [24].

To the best of our knowledge, no algorithm has been specifically designed for the special case of exact division.

Sparse divisibility testing. The problem of sparse divisibility testing is to determine, given two sparse polynomials F and G, whether G divides F. It is an open problem whether this problem admits a polynomial-time algorithm, that is an algorithm that runs in time $(T \log D)^{O(1)}$ where T bounds the number of nonzero terms of the inputs and D their degrees. We note that the division algorithms do not settle the problem. Indeed, the quotient of two sparse polynomials F and G can be exponentially larger than F and G.

The only general complexity result on this problem is due to Grigoriev, Karpinksi and Odlyzko [13] who show that the problem is in coNP under the Extended Riemann Hypothesis (ERH). Besides, the problem admits polynomial-time algorithms in the easy cases where $\deg(G)$, $\deg(F) - \deg(G)$ or $\#Q$ are polynomially bounded [30]. On the other hand, some related problems are coNP-hard, such as the divisibility of a product of sparse polynomials by a sparse polynomial, the computation of the constant coefficient of the quotient or the degree of the remainder [27].

1.2 Our contributions

We focus on the exact division of sparse polynomials. We first provide algorithms whose bit complexity are quasi-linear in the input and the output sparsities. Our algorithms work over finite fields and the integers, and are randomized. Over a finite field $\mathbb{F}_q$ of characteristic larger than the degree D of the inputs, it computes the quotient of two polynomials in $\tilde{O}_\epsilon(T \log D \log q)$ bit operations with probability at least $1-\epsilon$, where T bounds the number of nonzero terms of both the inputs and the output. For smaller characteristic, the complexity bound is $\tilde{O}_\epsilon(T \log^2 D(\log D + \log q))$. For polynomials over $\mathbb{Z}$ with coefficients bounded by C in absolute value, our algorithm performs $\tilde{O}_\epsilon(T(\log C + \log D \log S) + \log^3 S)$ bit operations where S is the maximum of D and the absolute value of the coefficients of the result. Our main technique is to adapt sparse polynomial interpolation algorithms to our needs. Our work focuses on the univariate case but it can be straightforwardly extended to the multivariate case using (randomized) Kronecker substitution [2]. We shall mention that the technique behind our exact division generalizes to the sparse interpolation of SLPs with divisions.

We also provide a polynomial time algorithm for special cases of the sparse polynomial divisibility testing problem when $\deg(F) = O(\deg(G))$. We prove that if G contains a small chunk of coefficients, with large *gaps* surrounding it, then one can test in polynomial time whether G divides F. More precisely, we require G to be written as $G_0 + X^k G_1 + X^\ell G_2$ with $\deg(G_1) = (T \log D)^{O(1)}$, $k - \deg(G_0) = \Omega(D)$ and $\ell - \deg(X^k G_1) = \Omega(D)$. This

We let $\tilde{O}(f(n)) = f(n)(\log f(n))^{O(1)}$ and $\tilde{O}_\epsilon(f(n)) = \tilde{O}(f(n)) \log^{O(1)} \frac{1}{\epsilon}$.

encompasses polynomials of the form $G_0+X^kG_1$ or $G_1+X^\ell G_2$. Our technique is to prove that in this situation, even if the quotient F/G may have an exponential number of nonzero terms, we are able to efficiently compute a multiple of the quotient that is sparse.

Notations. Let $F = \sum_{i=1}^{T} f_i X^{e_i}$. We use $\#F = T$ to denote its sparsity (number of nonzero terms) and $\mathrm{supp}(F) = \{e_1, \ldots, e_T\}$ for its support. If $F \in \mathbb{Z}[X]$, we use $\|F\|_\infty = \max_i |f_i|$ to denote its height. The *reciprocal* of F is the polynomial $F^\star = X^{\deg(F)}F(1/X)$.

2 Exact division

Our method to compute the exact quotient F/G of two sparse polynomials F and G relies on *sparse interpolation* algorithms. These algorithms usually take as input a straight-line program (SLP), or sometimes a blackbox, representing a sparse polynomial Q, together with bounds on the sparsity and the degree of Q. The output is the sparse polynomial Q given by the list of its nonzero monomials.

There are two main families of sparse polynomial interpolation algorithms. The first one, which originates with the work of Prony [28] and Ben-Or and Tiwari [6], uses evaluations of Q on geometric progressions. The second one, initiated by Garg and Schost [9], computes the reduction of Q *modulo* X^p-1 for some random prime p. This second line of work is more suitable to our case, to obtain the best complexity bounds.

For polynomials over $\mathbb{F}_q$, we rely on the best known sparse interpolation algorithms due to Huang [19] when $q > \deg Q$ and to Arnold, Giesbrecht and Roche [5] otherwise. These two algorithms compute the reduction of Q *modulo* X^p-1 for some random prime p. This computation is known as *SLP probing* and it can use dense polynomial arithmetic when p is small enough. The goal is then to reconstruct Q from $Q_p = Q \bmod X^p-1$. One difficulty comes from exponent recovery since a monomial cX^e of Q is mapped to $cX^{e \bmod p}$ in Q_p. A second one, called *exponent collision*, is when two distinct exponents e_1, $e_2 \in \mathrm{supp}(Q)$ are congruent *modulo* p. These collisions create the monomial $(c_1+c_2)X^e$ in Q_p, from which neither $c_1X^{e_1}$ nor $c_2X^{e_2}$ can be directly recovered.

The latter difficulty is handled similarly in [5] and [19]. Taking p at random in a large enough set of prime numbers, one can show that a substantial fraction of the monomials of Q do not collide during the reduction *modulo* X^p-1. Therefore, working with several random primes p allows the full reconstruction. The two algorithms mainly differ in the way they overcome the first difficulty.

Huang's very natural approach is to consider the derivative Q' of Q [19]. If the characteristic of $\mathbb{F}_q$ is larger than the degree of Q, a monomial cX^e of Q is mapped to ceX^{e-1} in Q'. Then it is mapped to $ceX^{(e-1) \bmod p}$ in $[Q']_p = Q' \bmod X^p-1$. Given an SLP for Q, one can efficiently compute an SLP for Q' using automatic differentiation [7]. If the monomial cX^e does not collide *modulo* X^p-1, it can be retrieved from its images in Q_p and $[Q']_p$ using a mere division on the coefficients.

With smaller characteristic, Huang's idea is no longer working since not all the integer exponents exist in $\mathbb{F}_q$. Instead, Arnold, Giesbrecht and Roche work *modulo* several primes p_i and use the Chinese remainder theorem to recover the exponent. For, they introduce the *diversification* technique to be able to match the corresponding monomials in $Q \bmod X^{p_i}-1$. Indeed, replacing Q with $Q(\alpha_j X)$ for several randomly chosen α_j's, the nonzero coefficients of $Q(\alpha_j X)$ are pairwise distinct with a good probability.

The main difficulty to adapt these approaches to the computation of $Q = F/G$ is that the division in $\mathbb{F}_q[X]/(X^p-1)$ is not well-defined. In the next section we show that taking α at random in a sufficiently large set is enough for $G(\alpha X)$ and $(\alpha X)^p-1$ to be coprime. We shall mention that this technique may be extended to other sparse polynomial interpolation algorithms. In particular, this includes slightly faster algorithms [18], but they rely on unproven *heuristics*.

2.1 Computation of a reduced quotient

Given F, $G \in \mathbb{F}_q[X]$ such that $F = GQ$, our aim is to compute $Q \bmod X^p-1 \in \mathbb{F}_q[X]$ for some prime p. Let $F_p = F \bmod X^p-1$, $G_p = G \bmod X^p-1$ and $Q_p = Q \bmod X^p-1$, then

$$F_p = G_pQ_p \bmod X^p-1. \tag{1}$$

If $\gcd(G_p, X^p-1) = 1$, then G_p is invertible *modulo* X^p-1, and Q_p can be computed. Otherwise, Equation (1) admits several solutions and does not define Q_p. The following lemma defines a probabilistic approach to overcome the latter difficulty.

Lemma 2.1. *Let A and $B \in \mathbb{F}_q[X]$ be two nonzero polynomials with $B(0) \neq 0$, and α randomly chosen in some extension $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$. Then $A(\alpha X)$ and $B(X)$ are coprime with probability at least $1 - \deg(A)\deg(B)/q^s$.*

Proof. Let β be a root of B in an algebraic closure $\overline{\mathbb{F}_q}$ of $\mathbb{F}_q$. Then β is a root of $A(\alpha X)$ if and only if $A(\alpha\beta) = 0$, that is α is a root of $A(\beta X)$. Since $A(\beta X)$ is nonzero and $\deg(A(\beta X)) = \deg(A)$, there exist at most $\deg(A)$ roots of $A(\beta X)$ in $\overline{\mathbb{F}_q}$. Since B has at most $\deg(B)$ roots in $\overline{\mathbb{F}_q}$, there are at most $\deg(A)\deg(B)$ values of α such that there exists a common root β of $A(\alpha X)$ and $B(X)$. Therefore, with probability at least $1 - \deg(A)\deg(B)/q^s$, $A(\alpha X)$ and $B(X)$ do not share a common root in $\overline{\mathbb{F}_q}$, that is they are coprime. □

Notations. *For $A \in \mathbb{F}_q[X]$, $\alpha \in \mathbb{F}_{q^s}$ and $p \geq 0$, let $A^{[\alpha]}$ be the polynomial $A(\alpha X)$, A_p be the polynomial $A(X) \bmod X^p - 1$ and $A_p^{[\alpha]}$ be the polynomial $A^{[\alpha]}(X) \bmod X^p - 1 = A(\alpha X) \bmod X^p - 1$.*

We remark that $A_p^{[\alpha]} \neq A_p(\alpha X)$. The idea is to apply Lemma 2.1 to G and $X^p - 1$. Instead of applying Equation (1) to F and G, we apply it to $F^{[\alpha]}$ and $G^{[\alpha]}$ to get $Q_p^{[\alpha]} = Q(\alpha X) \bmod X^p - 1$. In other words, we compute $Q_p^{[\alpha]}$ from the equation $F_p^{[\alpha]} = G_p^{[\alpha]} Q_p^{[\alpha]} \bmod X^p - 1$. If α is chosen at random in some extension $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$, $G^{[\alpha]}$ and $X^p - 1$ are coprime with probability at least $1 - p\deg(G)/q^s$ and $G_p^{[\alpha]}$ is invertible *modulo* $X^p - 1$ with the same probability. Since we compute $Q_p^{[\alpha]}$ for any α, we can adapt the algorithm of Arnold, Giesbrecht and Roche [5].

In order to adapt Huang's algorithm [19], we need to compute $Q'(X) \bmod X^p - 1$. To this end, we rely on the equality

$$[F']_p - [G']_p Q_p \bmod X^p - 1 = G_p [Q']_p \bmod X^p - 1 \tag{2}$$

where $[A']_p$ denotes $A'(X) \bmod X^p - 1$ for any $A \in \mathbb{F}_q[X]$. We notice that this equation is similar to Equation (1). Knowing Q_p, the equation defines $[Q']_p$ if and only if G_p is invertible *modulo* $X^p - 1$. This means that if α is chosen at random in $\mathbb{F}_{q^s}$, Equations (1) and (2) allow to compute both $Q_p^{[\alpha]}$ and $[(Q^{[\alpha]})']_p$ with probability at least $1 - p\deg(G)/q^s$, where $[(Q^{[\alpha]})']_p$ is the polynomial $(Q(\alpha X))' \bmod X^p - 1$. Next lemmas give the cost of these operations.

Lemma 2.2. *Let $A \in \mathbb{F}_q[X]$ of degree D, sparsity T and $\alpha \in \mathbb{F}_{q^s}$. Then $A^{[\alpha]}$ can be computed in $\tilde{O}(T \log Ds \log q)$ bit operations, and $A_p^{[\alpha]}$ in $O(T \log D \log\log p + Ts \log q)$ more bit operations.*

Proof. Computing $A^{[\alpha]} = A(\alpha X)$ requires T exponentiations of α, that is $O(T \log D)$ operations in $\mathbb{F}_{q^s}$, which gives a bit complexity of $\tilde{O}(T \log Ds \log q)$. Computing $A_p^{[\alpha]}$ from $A^{[\alpha]}$ requires T exponent divisions, that can be performed in $O(T \frac{\log D}{\log p})$ divisions on integers of size $\log p$, and $T - 1$ coefficient additions. □

Lemma 2.3. *Let F and $G \in \mathbb{F}_q[X]$ such that G divides F, and let $p \geq 0$ and $\alpha \in \mathbb{F}_{q^s}$ such that $G^{[\alpha]}$ and $X^p - 1$ are coprime. If $Q = F/G$, the polynomials $Q_p^{[\alpha]}$ and $[(Q^{[\alpha]})']_p$ can be computed in $\tilde{O}(T \log Ds \log q + ps \log q)$ bit operations, where $D = \deg(F)$ and T is a bound on the sparsities of F and G.*

Proof. To get $Q_p^{[\alpha]}$, the first step computes $F_p^{[\alpha]}$ and $G_p^{[\alpha]}$. Then we invert $G_p^{[\alpha]}$ *modulo* $X^p - 1$ using dense arithmetic and we multiply the result by $F_p^{[\alpha]}$. Then to get $[(Q^{[\alpha]})']_p$, we compute the derivatives of $F^{[\alpha]}$ and $G^{[\alpha]}$ and perform two more multiplications and one addition of dense polynomials, according to Equation (2). All dense polynomial operations cost $\tilde{O}(ps \log q)$ bit operations while the first step cost is given by Lemma 2.2. This concludes the proof since derivative cost is negligible. □

Huang's algorithm recovers monomials of Q from Q_p and $[Q']_p$. In our approach, we compute $Q_p^{[\alpha]}$ and $[(Q^{[\alpha]})']_p$ instead, and thus recover monomials of $Q^{[\alpha]}$ instead of Q. Yet, if cX^e is a monomial of $Q^{[\alpha]}$, the corresponding monomial in Q is $c\alpha^{-e}X^e$ and it can be computed in $\tilde{O}(\log(e)s \log q) = O(\log Ds \log q)$ bit operations.

2.2 Case of large characteristic

We first consider the case where the characteristic of $\mathbb{F}_q$ is larger than the degree D of the input polynomials. We begin with the main ingredient of Huang's algorithm to further adapt it to our needs. Recall that the idea is to recover Q from Q_p and $[Q']_p$.

Definition 2.4. *Let $Q \in \mathbb{F}_q[X]$ and p a prime number. Then* $\textsc{DLift}(Q_p, [Q']_p)$ *is the polynomial $\hat{Q} = \sum_e cX^e$ where the sum ranges over all the integers e such that Q_p contains the monomial $cX^{e \bmod p}$ and $[Q']_p$ contains the monomial $ceX^{(e-1) \bmod p}$.*

Clearly, if one knows both Q_p and $[Q']_p$, $\text{DLIFT}(Q_p, [Q']_p)$ can be computed in $\tilde{O}(p \log q)$ bit operations. Next lemma revamps the core of Huang's result [19]. Similar results are used in several interpolation algorithms [11, 4, 20].

Lemma 2.5. *Let $Q \in \mathbb{F}_q[X]$ of degree at most D and sparsity at most T. Let $p_1, \ldots, p_k$ be randomly chosen among the first N prime numbers, where $N = \max(1, \lceil 12(T-1)\log D \rceil)$. Let i that maximizes $\#Q_{p_i}$ and $\hat{Q} = \text{DLIFT}(Q_{p_i}, (Q')_{p_i})$. Then with probability at least $1 - 2^{-k}$, $\#(Q - \hat{Q}) \leq T/2$.*

The main idea in Huang's algorithm is to use this lemma $\log(T)$ times to recover all the coefficients of Q with probability at least $(1-2^{-k})^{\log T}$. To extend the algorithm to our case, we need to compute Q_p and $[Q']_p$ by choosing α in a suitable extension of $\mathbb{F}_q$ and compute $F(\alpha X)/G(\alpha X) \bmod X^p - 1$ as explained in Section 2.1. Next corollary establishes the size of that extension.

Corollary 2.6. *Let $G \in \mathbb{F}_q[X]$ of degree at most D, and α be a random element of $\mathbb{F}_{q^s}$ where $s = \lceil \log_q(\frac{965}{\epsilon} D^4) \rceil$. Then with probability at least $1-\epsilon$, $G(\alpha X)$ is coprime with $X^p - 1$ for each of the N first prime numbers, where N is defined as in Lemma 2.5.*

Proof. The polynomial $G(\alpha X)$ is coprime with all these polynomials if and only if it is coprime with their product. The degree of their product is the sum of the N first prime numbers, which is bounded by $N^2 \ln N$ (for $N > 3$). By Lemma 2.1, the probability that $G(\alpha X)$ be coprime with this product is at least $1 - DN^2 \ln N / q^s$ if α is chosen at random in $\mathbb{F}_{q^s}$. Since $s \geq \log_q(\frac{144}{\epsilon} D^4)$, $q^s \geq \frac{144}{\epsilon} D^4$. Furthermore, since $T \leq D$, $N \leq 12 D \log D$. This implies

$$\frac{DN^2 \ln N}{q^s} \leq \frac{144 D^3 \log^2(D) \ln(12 D \log D)}{\frac{965}{\epsilon} D^4} \leq \epsilon$$

since $144 \log^2(D) \ln(12 D \log D) \leq 965 D$ for all $D \geq 1$. □

By hypothesis on $\mathbb{F}_q$, we have $q \geq D$. This implies that $s = O(1)$ in Corollary 2.6 as long as $\frac{1}{\epsilon}$ remains polynomial in D. We now provide an algorithm for the exact division of sparse polynomials over large finite fields, given a bound on the sparsity of the quotient.

Algorithm 1 SPARSEDIVLARGECHARACTERISTIC

Input: $F, G \in \mathbb{F}_q[X]$ such that G divides F; a bound T on $\#F$, $\#G$ and $\#(F/G)$; $0 < \epsilon < 1$
Output: $F/G \in \mathbb{F}_q[X]$ with probability $\geq 1 - \epsilon$

1: Let $k = \lceil \log(\frac{2}{\epsilon} \log T) \rceil$ and $N = \max(1, \lceil 12(T-1)\log D \rceil)$
2: Compute the set $\mathscr{P}$ of the first N prime numbers
3: Compute an extension field $\mathbb{F}_{q^s}$ where $s = \lceil \log_q(\frac{1930}{\epsilon} D^4) \rceil$
4: Choose α at random in $\mathbb{F}_{q^s}$
5: Compute $F^{[\alpha]} = F(\alpha X)$ and $G^{[\alpha]} = G(\alpha X)$ and set $\hat{Q}^{[\alpha]} = 0$
6: **loop** $\lceil \log T \rceil$ times
7: Choose $p_1, \ldots, p_k$ at random in $\mathscr{P}$
8: **for** each p_i **do**
9: Compute $F^{[\alpha]}_{p_i}$, $G^{[\alpha]}_{p_i}$ ▷ Lemma 2.2
10: **if** $G^{[\alpha]}_{p_i}$ is not coprime with $X^{p_i} - 1$ **then return** *failure*
11: Compute $Q^{[\alpha]}_{p_i}$ and $[(Q^{[\alpha]})']_{p_i}$ ▷ Lemma 2.3
12: Let $p \in \{p_1, \ldots, p_k\}$ such that $\#Q^{[\alpha]}_p$ is maximal
13: Add $\text{DLIFT}(Q^{[\alpha]}_p - \hat{Q}^{[\alpha]}_p, [(Q^{[\alpha]})']_p - [(\hat{Q}^{[\alpha]})']_p)$ to $\hat{Q}^{[\alpha]}$
14: Return $\hat{Q}^{[\alpha]}(\alpha^{-1} X)$

Theorem 2.7. *Algorithm 1 is correct. It uses $\tilde{O}_\epsilon(T \log D \log q)$ bit operations.*

Proof. For the algorithm to succeed, $G(\alpha X)$ must be coprime with all the polynomials $X^p - 1$ used in the loop. By Corollary 2.6, it is coprime with all the polynomials $X^p - 1$ for $p \in \mathscr{P}$ with probability at least $1 - \frac{\epsilon}{2}$. Next, the algorithm succeeds if at each iteration, the number of monomials of $Q^{[\alpha]} - \hat{Q}^{[\alpha]}$ is halved. According to Lemma 2.5, this probability is at least $(1-2^{-k})^{\log T} \geq 1 - 2^{-k} \log T \geq 1 - \frac{\epsilon}{2}$. Therefore, the overall probability of success is at least $1 - \epsilon$.

Let us first note that $s \log q = O_\epsilon(\log q)$ since $D \leq q$. Step 2 takes $\tilde{O}(N) = \tilde{O}(T \log D)$ bit operations while step 3 takes $\tilde{O}(s^3 \log q) = \tilde{O}_\epsilon(\log q)$ bit operations. Computing the polynomials $F^{[\alpha]}$ and $G^{[\alpha]}$ costs

$\tilde{O}(T \log Ds \log q) = \tilde{O}_\epsilon(T \log D \log q)$ bit operations by Lemma 2.2. In the loop, as $p_i = O(N \log N)$, computing $F_{p_i}^{[\alpha]}$ and $G_{p_i}^{[\alpha]}$ costs $\tilde{O}_\epsilon(T(\log D + \log q))$ bit operations. Then operations on polynomials of degree at most p take $\tilde{O}(ps \log q)$ bit operations, that is $\tilde{O}_\epsilon(T \log D \log q)$ bit operations. Since this must be done for $\tilde{O}_\epsilon(\log T)$ primes in total, the overall cost of the algorithm is $\tilde{O}_\epsilon(T \log D \log q)$ bit operations. □

2.3 Case of small characteristic

When the field $\mathbb{F}_q$ has a characteristic smaller than the degree, Huang's technique is no more possible, and the best alternative is to use the algorithm of Arnold, Giesbrecht and Roche [5]. As mentioned before, their sparse interpolation algorithm computes $Q_p^{[\alpha]} = Q(\alpha X) \bmod X^p - 1$ for several values of α and p and they use the Chinese Remainder Theorem to recover the coefficients of Q. This is the last part of [5, procedure BuildApproximation], and we denote it by CRTLIFT. Next lemma summarizes their approach. It is the combination of [5, Lemma 3.1] for the value of λ, [5, Corollary 3.2] for γ and [5, Lemma 4.1] for m and s.

Lemma 2.8 ([5]). *Let $Q \in \mathbb{F}_q[X]$ a sparse polynomial of degree D and sparsity T. Let $0 < \mu < 1$, $\lambda = \max(21, \lceil \frac{40}{3}(T-1)\ln D \rceil)$, $\gamma = \lceil \max(8\log_\lambda D, 8\ln\frac{2}{\mu}) \rceil$, $m = \lceil \log\frac{1}{\mu} + 2\log(T(1+\frac{1}{2}\lceil \log_\lambda D \rceil)) \rceil$ and $s \geq \log_q(2D+1)$. Let $\hat{Q} = \text{CRTLIFT}((Q_{p_i}^{[\alpha_j]})_{ij})$ where $Q_{p_i}^{[\alpha_j]} = Q(\alpha_j X) \bmod X^{p_i} - 1$ for random primes $p_1, \ldots, p_\gamma$ in $]\lambda, 2\lambda[$ and random nonzero elements $\alpha_1, \ldots, \alpha_m$ of $\mathbb{F}_{q^s}$. Then with probability at least $1-\mu$, $\#(Q - \hat{Q}) \leq T/2$.*

In order to use such a lemma for sparse polynomial division, we need to compute $Q_{p_i}^{[\alpha_j]}$ for the γ primes p_i and the m points α_j. As explained in Section 2.1, $Q_{p_i}^{[\alpha_j]}$ can be efficiently computed as soon as $G(\alpha_j X)$ and $X^{p_i} - 1$ are coprime. To ensure, with good probability, that the coprimality property holds for every p_i and α_j, we choose the α_j's in a somewhat larger extension of $\mathbb{F}_q$. That is, we need to increase the bound on s given in Lemma 2.8 according to Lemma 2.1.

Lemma 2.9. *Let $G \in \mathbb{F}_q[X]$ of degree-D, $0 < \mu < 1$, λ, γ, m three integers and $p_1, \ldots, p_\gamma$ be prime numbers in $]\lambda, 2\lambda[$. Let $s = \lceil \log_q(2\frac{\lambda\gamma}{\mu} mD) \rceil$ and $\alpha_1, \ldots, \alpha_m$ be random elements of $\mathbb{F}_{q^s}$. Then with probability at least $1-\mu$, $G(\alpha_j)$ and $X^{p_i} - 1$ are coprime for all pairs (i, j), $1 \leq i \leq \gamma$ and $1 \leq j \leq m$.*

Proof. Let $\Pi = \prod_{i=1}^{\gamma} X^{p_i} - 1$. Its degree is $\sum_{i=1}^{\gamma} p_i \leq 2\lambda\gamma$. For any α_j, $G(\alpha_j X)$ is coprime with all the polynomials $X^{p_i} - 1$ if and only if $G(\alpha_j X)$ is coprime with Π. Since α_j is randomly chosen in $\mathbb{F}_{q^s}$ then by Lemma 2.1, the probability that $G(\alpha_j X)$ and Π are not coprime is at most $(2\lambda\gamma D)/q^s \leq \mu/m$ by definition of s. Therefore the probability that there is at least one α_j such that $G(\alpha_j X)$ and Π are not coprime is at most μ. □

Using Lemmas 2.8 and 2.9, we can adapt the sparse interpolation algorithm from [5] to the exact quotient of two sparse polynomials. It requires a larger extension than the original algorithm, but the growth is negligible by Lemma 2.8.

Algorithm 2 SPARSEDIVSMALLCHARACTERISTIC

Input: $F, G \in \mathbb{F}_q[X]$ such that G divides F; a bound T on $\#F$, $\#G$ and $\#(F/G)$; $0 < \epsilon < 1$
Output: $F/G \in \mathbb{F}_q[X]$ with probability $\geq 1 - \epsilon$

1: Let $\mu = \frac{\epsilon}{2\lceil \log T \rceil}$ and set λ, γ and m as in Lemma 2.8
2: Compute the set $\mathscr{P}$ of the prime numbers in $]\lambda, 2\lambda[$
3: Compute an extension field $\mathbb{F}_{q^s}$ where $s = \lceil \log_q(2\frac{\lambda\gamma}{\mu} mD) \rceil$
4: Set $\hat{Q} = 0$
5: **loop** $\lceil \log T \rceil$ times
6: Choose $p_1, \ldots, p_\gamma$ at random in $\mathscr{P}$
7: Choose $\alpha_1, \ldots, \alpha_m$ at random in $\mathbb{F}_{q^s}$
8: **for** each pair (p_i, α_j) **do**
9: Compute $F_{p_i}^{[\alpha_j]}$, $G_{p_i}^{[\alpha_j]}$ ▷ Lemma 2.2
10: **if** $G_{p_i}^{[\alpha_j]}$ is not coprime with $X^{p_i} - 1$ **then return** *failure*
11: Compute $Q_{p_i}^{[\alpha]}$ ▷ Lemma 2.3
12: Add $\text{CRTLIFT}((Q_{p_i}^{[\alpha_j]} - \hat{Q}_{p_i}^{[\alpha_j]})_{i,j})$ to $\hat{Q}$
13: Return $\hat{Q}$

Theorem 2.10. *Algorithm 2 is correct. It uses* $\tilde{O}_\epsilon(T \log^2 D(\log D + \log q))$ *bit operations where* $D = \deg(F)$.

Proof. The algorithm is a modification of [5, Procedure MajorityVoteSparseInterpolate]. The algorithm succeeds if at each iteration, every $G_{p_i}^{[\alpha_j]}$ is coprime with $X^{p_i} - 1$ and if CRTLIFT succeeds in recovering at least half of the terms of $Q - \hat{Q}$. Both conditions hold with probability $1 - \mu$ at each step. The global probability of success is thus at least $(1-\mu)^{2\lceil \log T \rceil} \geq 1 - 2\mu\lceil \log T \rceil \geq 1 - \epsilon$.

As in the original algorithm, the cost is dominated by the computation of all the $Q_{p_i}^{[\alpha_j]}$. There are $\gamma m \lceil \log T \rceil$ such polynomials to compute. Since $p_i < 2\lambda$ and $\alpha_j \in \mathbb{F}_{q^s}$, each computation costs $\tilde{O}((T \log D + \lambda) s \log q)$ bit operations according to Lemma 2.3. As $\lambda = O(T \log D)$, $\gamma = \tilde{O}_\epsilon(\log D)$, m is logarithmic and $s = \tilde{O}_\epsilon(1 + \log_q D)$, the algorithm requires $\tilde{O}_\epsilon(T \log^2 D(\log D + \log q))$ bit operations. □

2.4 Output sensitive algorithm

Both interpolation algorithms presented in the previous sections require a bound on the sparsity of the quotient. To overcome this difficulty, we use the same strategy as for sparse polynomial multiplication [12]. The idea is to guess the sparsity bound as we go using a fast verification procedure. For verifying exact quotient, we can directly reuse Algorithm VERIFYSP from [12] that verifies if $F = GQ$, with an error probability at most ϵ if $F \neq GQ$. It requires $\tilde{O}_\epsilon(T(\log D + \log q))$ bit operations over $\mathbb{F}_q$ and $\tilde{O}_\epsilon(T(\log D + \log C))$ bit operations over $\mathbb{Z}$ where C is a bound on the heights of F, G and Q.

Algorithm 3 SPARSEEXACTDIVISION

Input: F, $G \in \mathbb{F}_q[X]$ such that G divides F; $0 < \epsilon < 1$
Output: $F/G \in \mathbb{F}_q[X]$ with probability at least $1 - \frac{\epsilon}{2}$

1: $t \leftarrow 1$
2: **repeat**
3: $\quad t \leftarrow 2t$
4: $\quad$ Compute a tentative quotient $\hat{Q}$ using Algorithm 1 or 2, with sparsity bound t and probability $\frac{\epsilon}{2}$
5: **until** VERIFYSP$(F, G, \hat{Q}, \frac{\epsilon}{2t})$
6: **return** $\hat{Q}$

Theorem 2.11. *Let* F, $G \in \mathbb{F}_q[X]$ *such that* G *divides* F, $0 < \epsilon < 1$, $D = \deg(F)$ *and* $T = \max(\#F, \#G, \#(F/G))$. *With probability at least* $1 - \epsilon$, *Algorithm 3 returns* F/G *in* $\tilde{O}_\epsilon(T \log D \log q)$ *bit operations if* $\mathrm{char}(\mathbb{F}_q) > D$ *or* $\tilde{O}_\epsilon(T \log^2 D(\log D + \log q))$ *otherwise.*

Proof. The probability $1-\epsilon$ concerns both the correctness and the complexity of the algorithm. More precisely, we prove that the algorithm is correct with probability $\geq 1 - \frac{\epsilon}{2}$ and that it performs the claimed number of bit operations with probability $\geq 1 - \frac{\epsilon}{2}$.

The algorithm is incorrect when $F \neq G\hat{Q}$. This happens if at some iteration, Algorithm 1 or 2 returns an incorrect quotient but the verification algorithm fails to detect it. In other words, for the algorithm to return a correct answer, all the verifications must succeed. This happens with probability at least $1 - \sum_t \frac{\epsilon}{2t} \geq 1 - \frac{\epsilon}{2}$ since the sum ranges over powers of two.[†]

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 $2\#(F/G)$. If t attains this value, Algorithm 2.7 or 2.10 correctly computes F/G with probability at least $1 - \frac{\epsilon}{2}$ according to Theorems 1 and 2. That is, with probability at least $1 - \frac{\epsilon}{2}$, t is bounded by $2T$ and the number of iterations is $O(\log T)$. Depending on the characteristic, using Theorems 2.7 or 2.10 and the complexity of [12, Algorithm VERIFYSP], we obtain the claimed complexity with probability at least $1 - \frac{\epsilon}{2}$. □

2.5 Algorithm over the integers

For polynomials over $\mathbb{Z}[X]$, we cannot directly use Algorithm 3 with the variant of Huang's algorithm over $\mathbb{Z}$ [19]. Indeed, the coefficients arising during the computation may be dramatically larger than the inputs and the output. This is mostly due to the inversion of G *modulo* $X^p - 1$. Instead we use the standard technique that maps the computation over some large enough prime finite field. As we cannot tightly predict the size

[†]The error probability analysis of [12, Algorithm 2] is flawed and should be replaced by this new one.

of the coefficients, we can again reuse our guess-and-check approach with several prime numbers of growing size to discover it as we go. In order to use the fastest algorithm (Algorithm 1) we consider prime finite fields $\mathbb{F}_q$ such that q is larger than the input degree.

We first define a bound on the coefficient of the quotient of two sparse polynomial over $\mathbb{Z}[X]$ as the classic Mignotte's bound [10] on dense polynomial is too loose and it has no equivalent in the sparse case.

Lemma 2.12. *Let F, G, $Q \in \mathbb{Z}[X]$ be three sparse polynomials such that $F = QG$ and $T = \#Q$ is the number of nonzero coefficient of Q. Then*

$$\|Q\|_\infty \leq (\|G\|_\infty + 1)^{\lceil \frac{T-1}{2} \rceil} \|F\|_\infty.$$

Proof. Write $Q = \sum_{i=1}^{T} q_i X^{e_i}$ with $e_1 > e_2 > \cdots > e_T$. We use induction on the remainder and quotient sequence in the Euclidean division algorithm. Let $Q_j = \sum_{i=1}^{j} q_i X^{e_i}$ and $R_j = F - Q_j G$ be the elements of that sequence, starting with $R_0 = F$ and $Q_0 = 0$. The integer coefficients of Q are defined as $q_i = \mathsf{LC}(R_{i-1})/\mathsf{LC}(G)$ where LC denote the leading coefficient. We know from the algorithm that $R_i = R_{i-1} - q_i X^{e_i} G$ and $Q_i = Q_{i-1} + q_i X^{e_i}$. Since $R_0 = F$ and

$$\|R_i\|_\infty \leq \|R_{i-1}\|_\infty + |q_i| \times \|G\|_\infty \leq \|R_{i-1}\|_\infty (1 + \|G\|_\infty)$$

we have $\|R_i\|_\infty \leq \|F\|_\infty (1 + \|G\|_\infty)^i$. Since the reciprocal $Q^\star$ of Q is defined by the quotient $F^\star/G^\star$, we also get $\|R_i\|_\infty \leq \|F\|_\infty (1 + \|G\|_\infty)^{T-i}$. Therefore,

$$\|Q\|_\infty = \max_i(|q_i|) \leq \max_i \|R_i\|_\infty \leq \|F\|_\infty (1 + \|G\|_\infty)^{\lceil \frac{T-1}{2} \rceil}.$$ □

Algorithm 4 SPARSEEXACTDIVISIONOVERZ

Input: F, $G \in \mathbb{Z}[X]$ such that G divides F; $0 < \epsilon < 1$
Output: F/G with probability at least $1 - \frac{\epsilon}{2}$

1: Let $n = \deg(F)$, $i = 2$
2: **repeat**
3: $\quad i \leftarrow 2i$
4: $\quad$ Choose q at random in $]n, 2n[$, prime with prob. $\geq 1 - \frac{\epsilon}{2i}$
5: $\quad$ Compute the reductions $F_q = F \bmod q$ and $G_q = G \bmod q$
6: $\quad$ Compute $\hat{Q} = \text{SPARSEEXACTDIVISION}(F_q, G_q, \frac{\epsilon}{2i})$
7: $\quad n \leftarrow n^2$
8: **until** $\text{VERIFYSP}(F, G, \hat{Q}, \frac{\epsilon}{i})$
9: **return** $\hat{Q}$

Theorem 2.13. *Let F, G be two sparse polynomials in $\mathbb{Z}[X]$ such that G divides F, $0 < \epsilon < 1$, $D = \deg(F)$, $C = \max(\|F\|_\infty + \|G\|_\infty)$ and $T = \max(\#F, \#G, \#(F/G))$. With probability at least $1 - \epsilon$, Algorithm 4 returns F/G in $\tilde{O}_\epsilon(T(\log C + \log^2 D) + \log^3 D)$ bit operations if $D > 2\|F/G\|_\infty$ or $\tilde{O}_\epsilon(T(\log C + \log D \log \|F/G\|_\infty) + \log^3 \|F/G\|_\infty)$ bit operations otherwise.*

Proof. The proof goes along the same lines as for Theorem 2.11. With the same arguments, the probability that the algorithm returns an incorrect quotient is at most $\frac{\epsilon}{2}$.

In Step 4, we can use a Miller-Rabin based algorithm to compute a number q that is prime with probability at least $1 - \frac{\epsilon}{2i}$ in $\tilde{O}_{\epsilon/i}(\log^3 q)$ bit operations. Step 5 performs $O(T)$ divisions in $\tilde{O}(T \log C)$ bit operations. Thus by Theorem 2.11, an iteration of the loop correctly computes F_q/G_q in $\mathbb{F}_q[X]$ in $\tilde{O}_{\epsilon/i}(T \log D \log q + T \log C + \log^3 q)$ bit operations with probability at least $1 - \frac{\epsilon}{i}$. Let $Q = F/G$. As soon as $n > 2\|Q\|_\infty$, F_q/G_q is actually Q. Therefore, the algorithm stops with $n < 4\|Q\|_\infty^2$ with probability $\geq 1 - \frac{\epsilon}{i}$. If $D > 2\|Q\|_\infty$, q satisfies $2\|Q\|_\infty < q < 2D$ at the first iteration. Hence, the algorithm correctly computes Q in one iteration with probability $\geq 1 - \frac{\epsilon}{2}$. Its bit complexity is then $\tilde{O}_\epsilon(T \log^2 D + T \log C + \log^3 D)$.

Otherwise, at most $j = \lceil \log \log \|Q\|_\infty \rceil$ iterations are needed to get $2\|Q\|_\infty < n < q < 4\|Q\|_\infty^2$. Thus i is bounded by 2^{j+1} and Q is correctly computed in $\tilde{O}_{\epsilon/2^j}(T \log D \log \|Q\|_\infty + T \log C + \log^3 \|Q\|_\infty)$ bit operations with probability at least $1 - \sum_{k \geq 2} \frac{\epsilon}{2^k} \geq 1 - \frac{\epsilon}{2}$. By Lemma 2.12, $\|Q\|_\infty \leq (\|G\|_\infty + 1)^{\lceil \frac{T-1}{2} \rceil} \|F\|_\infty$, whence $\log \frac{2^j}{\epsilon} = O(\log T + \log \log C + \log \frac{1}{\epsilon})$ and $\tilde{O}_{\epsilon/2^j}(\cdot)$ is $\tilde{O}_\epsilon(\cdot)$.

In both cases, the cost of the loop body is as stated with probability at least $1 - \frac{\epsilon}{2}$. Since the verification with probability of success at least $1 - \frac{\epsilon}{2i}$ requires $\tilde{O}_{\epsilon/2i}(T(\log D + \log C + \log \|Q\|_\infty))$ bit operations and the maximal value of i is expected to be $O(\log \|Q\|_\infty)$, its cost is negligible compared to the loop body. Thus the algorithm works as stated with probability at least $1 - \epsilon$. □

3 Divisibility testing

Given two sparse polynomials F, $G \in \mathbb{K}[X]$, we want to check whether G divides F in polynomial time. If $\deg(F) = m+n-1$, $\deg(G) = m$ and $\#F$, $\#G \leq T$, then the input size is $O(T\log(m+n))$ and the divisibility check must cost $(T\log(m+n))^{O(1)}$. We first remind the only known positive results.

Proposition 3.1. *Let F, $G \in \mathbb{K}[X]$ of degrees $m+n-1$ and m respectively, and sparsity at most T. If either m or n is polynomial in $T\log(m+n)$, one can check whether G divides F in polynomial time.*

Proof. Let $F = GQ+R$ be the Euclidean division of F by G. When n is polynomially bounded, the Euclidean division algorithm runs in polynomial time and the verification is trivial. When m is polynomially bounded, the degree of the remainder is polynomial and it can be computed in polynomial time without computing Q. Indeed, it suffices to compute $X^e \bmod G$ for each exponent $e \in \mathrm{supp}(F)$ in polynomial time by fast exponentiation. □

The rest of the section can be seen as a generalization of the proposition. As long as one has a polynomial bound on the size of the quotient, the divisibility test is polynomial by either computing $F - GQ$ or asserting that $F = GQ$. We begin with a very simple remark, that we shall use repeatedly.

Remark 3.2. *Let F, $G \in \mathbb{K}[X]$, and $F^\star = X^{\deg F}F(1/X)$ and $G^\star = X^{\deg G}G(1/X)$ be their reciprocal polynomials. Then G divides F if and only if $G^\star$ divides $F^\star$. In this case, the quotient $Q^\star = F^\star \operatorname{quo} G^\star$ is the reciprocal of the quotient $Q = F \operatorname{quo} G$.*

Proof. From the definition, $(AB)^\star = A^\star B^\star$ for A, $B \in \mathbb{K}[X]$. Therefore, if there exists Q such that $F = GQ$, then $F^\star = G^\star Q^\star$. The converse follows since the reciprocal is involutive. □

We note that the equality $(F \operatorname{quo} G)^\star = F^\star \operatorname{quo} G^\star$ is not true in general when G does not divide F. Let $F = GQ+R$ with $\deg(R) < \deg(G)$. Then $F^\star = X^{\deg(F)}G(1/X)Q(1/X)+X^{\deg(F)}R(1/X) = G^\star Q^\star + X^{\deg(F)}R(1/X)$. Therefore $Q^\star = F^\star \operatorname{quo} G^\star$ if and only if $R = X^{\deg(F)}R(1/X)$.

3.1 Bounding the sparsity of the quotient

In this section, we provide a bound on the sparsity of the quotient $Q = F \operatorname{quo} G$, depending on the *gap* between the highest and second highest exponents in G. We make use of the following estimation.

Lemma 3.3. *Let $G \in \mathbb{K}[X]$ of degree m and sparsity $\#G$, such that $G = 1 - X^k G_1$ with $G_1 \in \mathbb{K}[X]$ of degree $m-k$. Then for all $t \geq 0$, $1/G \bmod X^{kt}$ has at most $\frac{1}{(t-1)!}(\#G+t-2)^{t-1}$ nonzero monomials.*

Proof. Since $G(0) = 1$, it is invertible in the ring of power series. Let $\phi = \sum_{i\geq 0} X^{ki}G_1^i \in \mathbb{K}[[X]]$ be its inverse. As soon as $i \geq t$, $X^{ki} \bmod X^{kt} = 0$. Therefore

$$\phi \bmod X^{kt} = \sum_{i=0}^{t-1} X^{ki}G_1^i \bmod X^{kt}.$$

Note that the support of G^{t-1} is a subset of $S = \{\sum_{j=1}^{t-1} e_j : e_j \in \mathrm{supp}(G)\}$ which has size at most $\binom{\#G+t-2}{t-1} \leq \frac{1}{(t-1)!}(\#G+t-2)^{t-1}$. Using the expansion $G^{t-1} = \sum_{i=0}^{t-1}\binom{t-1}{i}(-1)^i X^{ki}G_1^i$, one can see by identification that $\mathrm{supp}(\phi \bmod X^{kt}) \subseteq S$, whence the result. □

Corollary 3.4. *Let F and $G \in \mathbb{K}[X]$ of respective degrees $m+n-1$ and m, and respective sparsities $\#F$ and $\#G$. If $G = X^m - G_0$ with $G_0 \in \mathbb{K}[X]$ of degree $m-k$ then the quotient $Q = F \operatorname{quo} G$ has at most $\frac{1}{(\lceil n/k\rceil-1)!}\#F(\#G+\lceil n/k\rceil-2)^{\lceil n/k\rceil-1}$ nonzero monomials.*

Proof. Let $F = GQ+R$ with $\deg(R) < m$. It is classical that the reciprocal $Q^\star$ of Q equals $F^\star/G^\star \bmod X^n$ [10]. We can apply Lemma 3.3 to $G^\star$ since $G^\star = 1 - X^k G_0^\star$. Hence, $1/G^\star \bmod X^n$ has at most $\frac{1}{(\lceil n/k\rceil-1)!}(\#G-\lceil n/k\rceil-2)^{\lceil n/k\rceil-1}$ nonzero monomials, using $t = \lceil n/k\rceil$ and noting that $n \leq kt$. This implies that the sparsity of $Q^\star$, that is the sparsity of Q, is at most $\frac{1}{(\lceil n/k\rceil-1)!}\#F(\#G+\lceil n/k\rceil-2)^{\lceil n/k\rceil-1}$. □

Corollary 3.5. *Let F, $G \in \mathbb{K}[X]$ of respective degrees $m+n-1$ and m. If $G = 1 - X^k G_1$ and G divides F, then the quotient $Q = F \operatorname{quo} G$ has at most $\frac{1}{(\lceil n/k\rceil-1)!}\#F(\#G+\lceil n/k\rceil-2)^{\lceil n/k\rceil-1}$ nonzero monomials.*

Proof. We apply Corollary 3.4 to $F^\star$ and $G^\star$. Indeed, $G^\star = X^m - G_0$ for some G_0 of degree $m-k$ and since G divides F, we have $F \operatorname{quo} G = (F^\star \operatorname{quo} G^\star)^\star$. □

Next example shows that the bound does not hold anymore if G does not divide F.

Example 1. Let $F = X^{m+n-1} - 1$ and $G = X^m - X^{m-1} + 1$. Then $F \operatorname{quo} G = \sum_{i=0}^{n-1} X^i$ is as dense as possible.

If $F = GQ + R$ with some nonzero R then obviously $F - R = GQ$, that is G divides $F - R$. This implies that if R has few nonzero monomials, then Q as well since $F - R$ is a sparse polynomial. Conversely, if Q has few nonzero monomials, $R = F - GQ$ as well. As a result, we observe that the sparsities of the quotient and the remainder in the Euclidean division of F by $G = 1 + X^k G_1$ are polynomially related.

3.2 Algorithmic results

Let F, $G \in \mathbb{K}[X]$ of respective degrees $m + n - 1$ and m, with $n = O(m)$. Results of the previous section show that if $G = X^m - G_0$ with $\deg(G_0) \leq m - k$ for some $k = O(m)$, the sparsity of the quotient $F \operatorname{quo} G$ is polynomially bounded in the input size. If $G = 1 + X^k G_1$, the same holds for the quotient $F^\star \operatorname{quo} G^\star$. In both cases, this implies that one can check whether G divides F by a mere application of the Euclidean division algorithm. Our aim is to extend this approach to a larger family of divisors G through a generalization of Lemma 3.3. It is based on the following lemma.

Lemma 3.6. *Let F, G and $C \in \mathbb{K}[X]$, $C \neq 0$. Then G divides F if and only if G divides FC and C divides FC/G.*

Proof. If G divides F, then G clearly divides FC. Writing $F = GQ_1$, it is also clear that C divides $FC/G = Q_1 C$. Conversely, if G divides FC and C divides FC/G, we can write $FC/G = CQ_2$. Hence $F = GQ_2$ and G divides F. □

The generalization of Lemma 3.3 is given by the following lemma.

Lemma 3.7. *Let $G \in \mathbb{K}[X]$ of degree m and sparsity $\#G$, such that $G = G_0 - X^k G_1$ with $\deg(G_0) < k$ and $G(0) \neq 0$. Then for all t, $G_0^t/G \bmod X^{tk}$ has at most $\frac{1}{(t-1)!}(\#G + t - 2)^{t-1}$ nonzero monomials.*

Proof. Expanding $G_0/G = 1/(1 - X^k G_1 G_0^{-1}) = \sum_{i \geq 0} X^{ki} G_1^i G_0^{-i}$, we get $G_0^t/G = \sum_{i \geq 0} X^{ki} G_1^i G_0^{t-i-1}$ for all t. Since $X^{ki} \bmod X^{kt} = 0$ for $i \geq t$,

$$G_0^t/G \bmod X^{kt} = \sum_{i=0}^{t-1} X^{ki} G_1^i G_0^{t-1-i} \bmod X^{kt}.$$

The support of $G_0^t/G \bmod X^{kt}$ is also a subset of S defined in the proof of Lemma 3.3 since $G^{t-1} = \sum_{i=0}^{t-1} \binom{t-1}{i} (-1)^i X^{ki} G_1^i G_0^{t-1-i}$. Therefore, its sparsity is at most $\frac{1}{(t-1)!}(\#G + t - 2)^{t-1}$. □

Theorem 3.8. *Let F and $G \in \mathbb{K}[X]$ be two sparse polynomials, of degrees $m+n-1$ and m respectively, and sparsity at most T. One can check whether G divides F in polynomial time if $G = G_0 - X^k G_1$ where $k - \deg(G_0) = \Omega(n)$ and either $\deg(G_0)$ or $\deg(G_1)$ is bounded by a polynomial function of the input size.*

Proof. We first note that we can first remove any power of X that divides F or G. If X^a divides F and X^b divides G, then G divides F if and only if $b \leq a$ and G/X^b divides F/X^a. Therefore, we assume from now on that $G(0)$ and $F(0)$ are nonzero. This implies in particular that G and G_0 are both invertible in the ring of power series over $\mathbb{K}$. We treat the case $\deg(G_0) = (T \log(m+n))^{O(1)}$. The second case is directly obtained by taking reciprocals.

By Lemma 3.6, for any integer $t \geq 0$, G divides F if and only if G divides $G_0^t F$ and G_0^t divides FG_0^t/G. Our algorithm checks these conditions for some t such that $k - \ell \geq n/t$, where $\ell = \deg(G_0)$.

By Lemma 3.7, $G_0^t/G \bmod X^{kt}$ has at most $\frac{1}{(t-1)!}(T + t - 2)^{t-1}$ nonzero terms, whence $FG_0^t/G \bmod X^{kt}$ at most $\frac{1}{(t-1)!} T(T+t-2)^{t-1}$. Note that $t = O(1)$ since $k - \ell = \Omega(n)$, and that $kt \geq n + \ell t$. Since $G_0^t/G \bmod X^{n+\ell t} = ((FG_0^t)^\star \operatorname{quo} G^\star)^\star$, the sparsity of $(FG_0^t)^\star \operatorname{quo} G^\star$ is at most $T^{O(1)}$. One can compute this quotient and check whether the remainder vanishes to test in polynomial time if G divides FG_0^t. If the test fails, G does not divide F. Otherwise, we have computed a polynomial Q_0 such that $FG_0^t = Q_0 G$. It remains to check whether G_0^t divides Q_0. Proposition 3.1 provides a polynomial-time algorithm for this since $\deg(G_0^t)$ is polynomially bounded. □

The previous proof extends to more general divisors. It only requires a polynomial bound on the sparsity of Q_0 and a polynomial-time algorithm to test whether G_0^t divides Q_0. The second step can be a recursive call if G_0^t satisfies the conditions in the theorem. This provides the following generalization of the theorem.

Corollary 3.9. *Let F and $G \in \mathbb{K}[X]$ be two sparse polynomials, of degrees $m+n-1$ and m respectively, and sparsity at most T. One can check whether G divides F in polynomial time if $G = G_0 + X^k G_1 - X^\ell G_2$ with G_0, G_1, $G_2 \in \mathbb{K}[X]$ such that $k - \deg(G_0)$ and $\ell - k - \deg(G_1)$ are both $\Omega(n)$ and $\deg(G_1) = (T\log(m+n))^{O(1)}$.*

Proof. We assume that $T\log(m+n) = n^{o(1)}$. Otherwise, one can use Proposition 3.1. Using Lemma 3.7, $F(G_0 + X^k G_1)^t / G \bmod X^{kt}$ has at most $T^{O(1)}$ nonzero monomials for $t = O(1)$. Therefore, as previously, we can compute the quotient $(F(G_0 + X^k G_1)^t)^\star \operatorname{quo} G^\star$ for $t = \lceil n/(\ell - k - \deg(G_1))\rceil$, in polynomial time. If the remainder is nonzero, G does not divide F. Otherwise, we have computed a polynomial Q_{01} such that $F(G_0+X^k G_1)^t = Q_{01} G$. It remains to test whether $H = (G_0+X^k G_1)^t$ divides Q_{01}. We show that the polynomial H satisfies the conditions of Theorem 3.8. Let us write

$$H = \sum_{i=0}^{t} \binom{t}{i} X^{ki} G_1^i G_0^{t-i} = X^{kt} G_1^t + \sum_{i=0}^{t-1} \binom{t}{i} X^{ki} G_1^i G_0^{t-i} = X^{kt} G_1^t + H_0$$

where H_0 has degree at most $k(t-1) + \deg(G_1)(t-1) + \deg(G_0)$. Then $kt - \deg(H_0) \geq k - \deg(G_0) - (t-1)\deg(G_1) = \Omega(n)$ since $k - \deg(G_0) = \Omega(n)$ and $\deg(G_1) = (T\log(m+n))^{O(1)} = n^{o(1)}$. One can test whether H divides Q_{01} in polynomial-time using Theorem 3.8. □

Theorem 3.8 and Corollary 3.9 cover cases were the quotient of the polynomials and the quotient of their reciprocals are both dense, as shown in the following example.

Example 2. Let $F = X^{2n-1} - X^n - X^{n-1} + X^2 - X + 1$ and $G = G_0 + X^{n-1} G_1$ where $G_0 = G_1 = 1 - X$. Then $F \operatorname{quo} G = \sum_{i=0}^{n-1} X^i$ and $F^\star \operatorname{quo} G^\star = X^{n-1} + \sum_{i=0}^{n-3} X^i$.

Acknowledgments

We are grateful to the reviewers for their insightful comments.

References

[1] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster sparse multivariate polynomial interpolation of straight-line programs. *J. Symb. Comput.*, 75:4–24, 2016. DOI:10.1016/j.jsc.2015.11.005.

[2] A. Arnold and D. S. Roche. Multivariate sparse interpolation using randomized Kronecker substitutions. In *ISSAC'14*, pages 35–42, 2014. DOI:10.1145/2608628.2608674.

[3] A. Arnold and D. S. Roche. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *ISSAC'15*, pages 29–36, 2015. DOI:10.1145/2755996.2756653.

[4] A. Arnold, M. Giesbrecht, and D. S. Roche. Faster Sparse Interpolation of Straight-Line Programs. In *CASC'13*, pages 61–74, 2013. DOI:10.1007/978-3-319-02297-0_5.

[5] A. Arnold, M.. Giesbrecht, and D. S. Roche. Sparse interpolation over finite fields via low-order roots of unity. In *ISSAC'14*, pages 27–34, 2014. DOI:10.1145/2608628.2608671.

[6] M. Ben-Or and P. Tiwari. A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. In *STOC'88*, pages 301–309, 1988. DOI:10.1145/62212.62241.

[7] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.

[8] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC'02*, pages 592–601, 2002. DOI:10.1145/509907.509992.

[9] S. Garg and É. Schost. Interpolation of polynomials given by straight-line programs. *Theor. Comput. Sci.*, 410(27):2659–2662, 2009. DOI:10.1016/j.tcs.2009.03.030.

[10] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3rd edition, 2013.

[11] M. Giesbrecht and D. S. Roche. Diversification improves interpolation. In *ISSAC'11*, pages 123–130, 2011. DOI:10.1145/1993886.1993909.

[12] P. Giorgi, B. Grenet, and A. Perret du Cray. Essentially optimal sparse polynomial multiplication. In *ISSAC'20*, pages 202–209, 2020. DOI:10.1145/3373207.3404026.

[13] D. Grigoriev, M. Karpinski, and A. M. Odlyzko. Short proofs for nondivisibility of sparse polynomials under the extended riemann hypothesis. *Fund. Inform.*, 28(3-4):297–301, 1996.

[14] J. van der Hoeven. Probably faster multiplication of sparse polynomials. preprint, 2020. URL: https://hal.archives-ouvertes.fr/hal-02473830.

[15] J. van der Hoeven and G. Lecerf. On the Complexity of Multivariate Blockwise Polynomial Multiplication. In *ISSAC'12*, pages 211–218, 2012. DOI:10.1145/2442829.2442861.

[16] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *J. Symb. Comput.*, 50:227–254, 2013. DOI:10.1016/j.jsc.2012.06.004.

[17] J. van der Hoeven and G. Lecerf. Sparse Polynomial Interpolation in Practice. *ACM Commun. Comput. Algebra*, 48(3/4):187–191, 2014. DOI:10.1145/2733693.2733721.

[18] J. van der Hoeven and G. Lecerf. Sparse polynomial interpolation. Exploring fast heuristic algorithms over finite fields. preprint, 2019. URL: https://hal.archives-ouvertes.fr/hal-02382117.

[19] Q. Huang. Sparse polynomial interpolation over fields with large or zero characteristic. In *ISSAC'19*, pages 219–226, 2019. DOI:10.1145/3326229.3326250.

[20] Q. Huang and X. Gao. Faster interpolation algorithms for sparse multivariate polynomials given by straight-line programs. *J. Symb. Comput.*, 101:367–386, 2020. DOI:10.1016/j.jsc.2019.10.005.

[21] S. C. Johnson. Sparse polynomial arithmetic. *ACM SIGSAM Bulletin*, 8(3):63–71, 1974. DOI:10.1145/1086837.1086847.

[22] E. Kaltofen and W.-s. Lee. Early termination in sparse interpolation algorithms. *J. Symb. Comput.*, 36(3):365–400, 2003. DOI:10.1016/S0747-7171(03)00088-9.

[23] M. Monagan and R. Pearce. Parallel sparse polynomial multiplication using heaps. In *ISSAC'09*, pages 263–270, 2009. DOI:10.1145/1576702.1576739.

[24] M. Monagan and R. Pearce. Sparse polynomial division using a heap. *J. Symb. Comput.*, 46(7):807–822, 2011. DOI:10.1016/j.jsc.2010.08.014.

[25] M. Monagan and R. Pearce. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. In *CASC'07*, pages 295–315, 2007.

[26] V. Nakos. Nearly optimal sparse polynomial multiplication. *IEEE T. Inform. Theory*, 66(11):7231–7236, 2020. DOI:10.1109/TIT.2020.2989385.

[27] D. A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theor. Comput. Sci.*, 31(1):125–138, 1984. DOI:10.1016/0304-3975(84)90130-0.

[28] R. Prony. Essai expérimental et analytique sur les lois de la dilatabilité de fluides élastique et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures. *J. École Polytechnique*, 1(Floréal et Prairial III):24–76, 1795. URL: https://gallica.bnf.fr/ark:/12148/bpt6k433661n/f32.item.

[29] D. S. Roche. Chunky and equal-spaced polynomial multiplication. *J. Symb. Comput.*, 46(7):791–806, 2011. DOI:10.1016/j.jsc.2010.08.013.

[30] D. S. Roche. What can (and can't) we do with sparse polynomials? In *ISSAC'18*, pages 25–30, 2018. DOI:10.1145/3208976.3209027.

[31] T. Yan. The Geobucket Data Structure for Polynomials. *J. Symb. Comput.*, 25(3):285–293, 1998. DOI:10.1006/jsco.1997.0176.