Essentially optimal sparse polynomial multiplication

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

January 31, 2020

Abstract

In this article, we present a probabilistic algorithm to compute the product of two univariate sparse polynomials over a field with a number of bit operations that is quasi-linear in the size of the input and the output. Our algorithm works for any field of characteristic zero or larger than the degree. We mainly rely on sparse interpolation and on a new algorithm for verifying a sparse product that has also a quasi-linear time complexity. Using Kronecker substitution techniques we are able to extend our result to the multivariate case.

1 Introduction

Polynomial is one of the most basic object in computer algebra and the study of fast polynomial operations still remains one of the most challenging task. The most classical way of dealing with polynomial is to either use the dense representation that stores all the coefficients in a vector or the sparse one that keeps only the nonzero monomials. While quasi-optimal algorithms have been designed since decades for the dense representation, this not yet the case for the more compact sparse representation, mainly due to the specificity of the representation.

In the sparse representation, a polynomial $F = \sum_{i=0}^{D} f_i X^i \in R[X]$ is represented as a list of pairs (e_i, f_{e_i}) such all the f_{e_i}'s are nonzero. If s is a bound on the size of the coefficients of F, T its *sparsity* (that is its number of nonzero coefficients) and D its degree, the size of the representation of F is $\mathcal{O}(T(s + \log D))$ bits. It is classical to consider $s = \mathcal{O}(\log(\|F\|_\infty))$ when $R = \mathbb{Z}$ and $s = \mathcal{O}(\log(q))$ when $R = \mathbb{F}_q$. This representation naturally extends to multivariate polynomials, where each exponent is replaced by a vector of exponents, thus giving a size of $\mathcal{O}(T(s + n \log D)$ where n is the number of variables.

Several problems have been investigated to design fast algorithms for sparse polynomials, including arithmetic operations, interpolation and factorization (we refer to the excellent survey of Roche and the references therein [19]). Note that *fast* algorithms for sparse polynomials must have a logarithmic (or poly-logarithmic) dependency on the degree, contrary to the dense case. Unfortunately, as shown by several NP-hardness results, such fast algorithms might not even exist unless $\mathsf{P} \neq \mathsf{NP}$. This is for instance the case for GCD computations [16].

In this paper, we are interested in a more basic problem that is the product of two polynomials for which we provide the first algorithm being quasi-optimal in the input and the output size.

1.1 Previous work

The main difficulty and most interesting aspect of the sparse polynomial multiplication is the fact that the size of the output does not exclusively depend on the size of the inputs, contrary to the dense case. Indeed, the product of two polynomials F and G has at most $\#F\#G$ nonzero coefficients (where $\#F$ and $\#G$ denote their respective sparsities), but it may have as few as 2 nonzero coefficients.

Example 1. Let $F = X^{14} + 2X^7 + 2$, $G = 3X^{13} + 5X^8 + 3$ and $H = X^{14} - 2X^7 + 2$. Then $FG = 3X^{27} + 5X^{22} + 6X^{20} + 10X^{15} + 3X^{14} + 6X^{13} + 10X^8 + 6X^7 + 6$ has nine terms, while $FH = X^{28} + 4$ has only two.

Given two polynomials of sparsity T, their product can be computed by generating the T^2 possible monomials, sorting them by increasing degree and merging the monomials of the same degree. Using radix sort, this algorithm takes $\mathcal{O}(T^2(\mathsf{M}_R + \log D))$ bit operations, where M_R denotes the cost of one operation in R. A major drawback of this approach is its space complexity that exhibits a T^2, even if the result has less T^2

terms. Many improvements have been proposed to either lower down this space complexity while keeping the same time complexity or to extend this approach to the multivariate polynomials case and then to provide fast implementations in practice [12, 13, 14]. While all these results mainly improved the space complexity, none of them reduces the T^2 terms appearing in the time complexity.

Letting apart space complexity consideration, some other works have made the time complexity to be more output dependent when the inputs are sufficiently structured to know in advance that the output size will be small [18], especially in the multivariate case [23, 22], or when the support of the output is known in advance [9].

Since the output size could be quadratic in T in the worst case, no improvement should be expected in terms of the inputs size only. But there could be less than T^2 terms in the output for two reasons. First, there may be some exponent *collisions*. If $F = \sum_{i=1}^{T} f_i X^{\alpha_i}$ and $G = \sum_{j=1}^{T} g_j X^{\beta_j}$, there may exist two distinct pairs of indices (i_1, j_1) and (i_2, j_2) such that $\alpha_{i_1} + \beta_{j_1} = \alpha_{i_2} + \beta_{j_2}$. Such collisions decrease the number of terms of the result. The second source of decrease comes from the coefficients. In the previous case, the resulting monomial is $(f_{i_1} g_{j_1} + f_{i_2} g_{j_2}) X^{\alpha_{i_1} + \beta_{j_1}}$ which could be zero depending on the coefficient values.

In [2] Arnold and Roche take into account the exponent collision by defining the *structural support* and the *structural sparsity* of a sparse polynomial product FG. The structural support of FG is the sumset of the exponents appearing in F and G that is $\{\alpha_i + \beta_j : 1 \leq i, j \leq T\}$ and the structural sparsity its size. Let $F, G \in \mathbb{Z}[X]$ with coefficient of bitsize at most B, $H = FG$ and S be the structural sparsity of FG. Arnold and Roche's algorithm computes H in $\tilde{\mathcal{O}}(S \log D + \#HB)$ bit operations. They also extend their algorithm to work over any finite fields and also to the multivariate polynomials case. Actually, Cole and Hariharan described an algorithm requiring $\tilde{\mathcal{O}}(S \log^2 D + \log^3 D)$ bit operations for polynomials with nonnegative integer coefficients of bitsize $\mathcal{O}(\log D)$ in the context of sparse wildcard matching [5].

Note that Arnold and Roche's algorithm is quasi-optimal when the structural sparsity is close, or even equal, to the sparsity of the result. Let $H = FG$ where F and G are two polynomials of sparsity $T \geq 2$, and S be the structural sparsity of FG. Then by definition we have $2 \leq \#H \leq S \leq T^2$. Even though S and $\#H$ can be close, their difference can reach $\mathcal{O}(T^2)$ as shown by the next example, which means that their algorithm can be far from the optimal.

Example 2. Let $F = \sum_{i=0}^{T-1} X^i$, $G = \sum_{i=0}^{T-1} (X^{Ti+1} - X^{Ti})$ and $H = FG$. Then $\#F = T$, $\#G = 2T$ and the structural sparsity of FG is $T^2 + 1$ while $H = X^{T^2} - 1$ has sparsity 2.

An attempt to avoid the dependency on the structural sparsity in the complexity for the case of integer polynomials is presented in [15], in the word RAM model with large word size. Nevertheless, the algorithm translated to a more standard bit complexity model has non quasi-linear complexity $\tilde{\mathcal{O}}((\#H \log^2 D + \log^3 D)(\log D + B))$, and it always returns polynomials with nonnegative coefficients.

While in the dense case the quasi-optimal algorithms rely on an evaluation-interpolation scheme, this is not yet the case for the fastest one in the sparse setting. Even if sparse interpolation has received a lot of attention, from the early results of Prony [17] and Ben-Or and Tiwari [4] to the most recent result of Huang [10], it does not resemble to the dense interpolation problem. Nevertheless, Arnold explores in his PhD thesis the use of sparse interpolation to perform sparse polynomial multiplication [1] and it reveals fruitful in obtaining the algorithm with Roche in [2]. Our work continues this approach a step further and we show that sparse interpolation can indeed be used to obtain a quasi-optimal multiplication algorithm for sparse polynomials.

1.2 Our contributions

Our main result is summarized in Theorem 1.1. Here, and throughout the article, $\tilde{\mathcal{O}}(f(n))$ denotes $\mathcal{O}(f(n) \log^{k_1}(f(n)))$ for some constant $k_1 > 0$. We extend the complexity notations to $\mathcal{O}_\epsilon$ and $\tilde{\mathcal{O}}_\epsilon$ for hiding some polynomial factors in $\log(\frac{1}{\epsilon})$. For a polynomial $F = \sum_{i=1}^{T} f_i X^{e_i}$, we write $\|F\|_\infty = \max_i |f_i|$ for its height, $\#F$ for its number of nonzero terms and $\mathrm{supp}(F) = \{e_1, \ldots, e_T\}$.

Theorem 1.1. *Given two sparse polynomials F and G over $\mathbb{Z}$, Algorithm* SparseProduct *computes $H = FG$ in $\tilde{\mathcal{O}}_\epsilon(T(\log D + \log C))$ bit operations with probability at least $1 - \epsilon$, where $D = \deg(H)$, $C = \max(\|F\|_\infty, \|G\|_\infty, \|H\|_\infty)$ and $T = \max(\#F, \#G, \#H)$. The algorithm extends naturally to finite fields with characteristic larger than D with the same complexity where C denotes the cardinality.*

This result is based on two main ingredients. On the one hand, we adapt Huang's sparse interpolation algorithm [10] to be able to interpolate $F \times G$ in quasi-linear time. Note that the original algorithm does not reach such complexity. On the other hand, any sparse interpolation algorithms, including Huang's one, require a bound on the sparsity of the result. We circumvent the latter difficulty by designing a quasi-linear

time verification algorithm for sparse product which allows us to replace such an *a priori* knowledge by an *a posteriori* check.

Theorem 1.2. *Given three sparse polynomials F, G and H over $\mathbb{F}_q$ or $\mathbb{Z}$, Algorithm* VERIFYSP *tests whether $FG = H$ in $\tilde{\mathcal{O}}_\epsilon(T(\log D + B))$ bit operations, where $D = \deg(H)$, B is a bound on the bitsize of the coefficients of F, G and H, and $T = \max(\#F, \#G, \#H)$. The answer is always correct if $FG = H$, and the probability of error is at most ϵ otherwise.*

Finally, we show that using Kronecker substitution our sparse polynomial multiplication algorithm also extends to the multivariate case with a quasi-linear bit complexity $\tilde{\mathcal{O}}_\epsilon(T(n \log d + B))$ where n is the number of variables and d is the maximal partial degree on each variable. Nevertheless, over finite fields our approach requires an exponentially large characteristic. Using the randomized Kronecker substitution [3] we derive a fast algorithm that works for characteristic that remains polynomial in the input size. Though not quasi-linear our bit complexity is $\tilde{\mathcal{O}}_\epsilon(nT(\log d + B))$ and it improves the known complexity for this case.

2 Preliminaries

We denote by $\mathsf{I}(n) = \mathcal{O}(n \log n)$ the bit complexity of the multiplication of two integers of at most n bits [8]. Similarly, we denote by $\mathsf{M}_q(D) = \mathcal{O}(D \log q \log(D \log q) 4^{\log^* D})$ the bit complexity of the multiplication of two *dense* polynomials of degree at most D over $\mathbb{F}_q$ where q is prime [7]. The cost of multiplying two elements of $\mathbb{F}_{q^s}$ is $\mathsf{M}_q(s)$. The cost of multiplying two dense polynomials over $\mathbb{Z}$ of heights at most C and degrees at most D is $\mathsf{M}_{\mathbb{Z}}(D, C) = \mathsf{I}(D(\log C + \log D))$ [24, Chapter 8].

Since our algorithms use reductions *modulo* $X^p - 1$ for some prime number p, we first review useful related results.

Theorem 2.1 ([20])**.** *If $\lambda \geq 21$, there are at least $\frac{3}{5}\lambda / \ln \lambda$ prime numbers in $[\lambda, 2\lambda]$.*

Proposition 2.2. *There exists an algorithm* RANDOMPRIME(λ, ϵ) *that returns an integer p in $[\lambda, 2\lambda]$, such that p is prime with probability at least $1 - \epsilon$. Its bit complexity is $\tilde{\mathcal{O}}_\epsilon(\log^k \lambda)$ for some $k \leq 7$.*

We need two distinct properties for the reductions *modulo* $X^p - 1$. The first one is classical in sparse interpolation to bound the probability of exponent collision in the residue (see [2, Lemma 3.3]).

Proposition 2.3. *Let H be a polynomial of degree at most D and sparsity at most T, $0 < \epsilon < 1$ and $\lambda = \max(21, \frac{10}{3\epsilon} T^2 \ln D)$. Then with probability at least $1 - \epsilon$,* RANDOMPRIME$(\lambda, \frac{\epsilon}{2})$ *returns a prime number p such that $H \bmod X^p - 1$ has the same number of terms as H, that is no collision of exponents occurs.*

The second property allows to bound the probability that a polynomial vanishes *modulo* $X^p - 1$.

Proposition 2.4. *Let H be a nonzero polynomial of degree at most D and sparsity at most T, $0 < \epsilon < 1$ and $\lambda = \max(21, \frac{10}{3\epsilon} T \ln D)$. Then with probability at least $1 - \epsilon$,* RANDOMPRIME$(\lambda, \frac{\epsilon}{2})$ *returns a prime number p such that $H \bmod X^p - 1 \neq 0$.*

Proof. It is sufficient, for $H \bmod X^p - 1$ to be nonzero, that there exist one exponent e of H that is not congruent to any other exponents e_j modulo p. In other words, it is sufficient that p does not divide any of the $T - 1$ differences $\delta_j = e_j - e$.

Noting that $\delta_j \leq D$, the number of primes in $[\lambda, 2\lambda]$ that divide at least one δ_j is at most $\frac{(T-1)\ln D}{\ln \lambda}$. Since there exists $\frac{3}{5}\lambda / \ln \lambda$ primes in this interval, the probability that a prime randomly chosen from it divides at least one δ_j is at most $\epsilon/2$. RANDOMPRIME$(\lambda, \epsilon/2)$ returns a prime in $[\lambda, 2\lambda]$ with probability at least $1 - \epsilon/2$, whence the result. □

The following two propositions will be useful to either reduce integer coefficients modulo some prime number or to construct an extension field.

Proposition 2.5. *Let $H \in \mathbb{Z}[X]$ be a nonzero polynomial, $0 < \epsilon < 1$ and $\lambda \geq \max(21, \frac{10}{3\epsilon} \ln \|H\|_\infty)$. Then with probability at least $1 - \epsilon$,* RANDOMPRIME$(\lambda, \frac{\epsilon}{2})$ *returns a prime q such that $H \bmod q \neq 0$.*

Proof. Let h_i be a nonzero coefficient of H,

a random prime from $[\lambda, 2\lambda]$ divides h_i with probability at most $\frac{5}{3} \ln \|H\|_\infty / \lambda \leq \epsilon/2$. Since RANDOMPRIME$(\lambda, \epsilon/2)$ returns a prime in $[\lambda, 2\lambda]$ with probability at least $1 - \epsilon/2$ the result follows. □

Proposition 2.6 ([21, Chapter 20])**.** *There exists an algorithm that, given a finite field $\mathbb{F}_q$, an integer s and $0 < \epsilon < 1$, computes a degree-s polynomial in $\mathbb{F}_q[X]$ that is irreducible with probability at least $1 - \epsilon$. Its bit complexity is $\tilde{\mathcal{O}}_\epsilon(s^3 \log q)$.*

3 Sparse polynomial product verification

Verifying a product $FG = H$ of dense polynomials over an integral domain R simply falls down to testing $F(\alpha)G(\alpha) = H(\alpha)$ for some random point $\alpha \in R$. This approach exhibits an optimal linear number of operations in R but it is not deterministic (optimal deterministic algorithm does not yet exists). When $R = \mathbb{Z}$ or $\mathbb{F}_q$, the complexity becomes quasi-linear in the input size as the best algorithm using a divide and conquer approach requires $\tilde{\mathcal{O}}(DB)$ where B bounds the bitsize of the coefficients.

For sparse polynomials with T nonzero coefficients, evaluation is not quasi-linear since the input size is only $\mathcal{O}(T(\log D + B))$. Indeed, computing α^D requires $\mathcal{O}(\log D)$ operations in R which implies a bit complexity of $\tilde{\mathcal{O}}(\log D \log q)$ when $R = \mathbb{F}_q$. Applying this computation to the T non zero monomial gives a bit complexity of $\tilde{\mathcal{O}}(T \log D \log q)$. We shall mention that the latter approach can be improved to $\tilde{\mathcal{O}}(\log D \log q\ (1 + T/\log\log D))$ using Yao's result [25] on simultaneous exponentiation. When $R = \mathbb{Z}$ the complexity is even worse as α^D has size $\mathcal{O}(D)$.

Our approach to obtain a quasi-linear complexity is to perform the evaluation *modulo* $X^p - 1$ for some random prime p. This requires to evaluate the polynomial $[(FG) \bmod X^p - 1]$ on α without computing it.

3.1 Modular product evaluation

Lemma 3.1. *Let F and G be two sparse polynomials in $R[X]$ with $\deg F, \deg G \leq p - 1$ and $\alpha \in R$. Then $(FG) \bmod X^p - 1$ can be evaluated on α using $\mathcal{O}((\#F + \#G)\log p)$ operations in R.*

Proof. Let $H = (FG) \bmod X^p - 1$. The computation of H corresponds to the linear map

$$\underbrace{\begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{p-1} \end{pmatrix}}_{\vec{h}} = \underbrace{\begin{pmatrix} f_0 & f_{p-1} & \cdots & f_1 \\ f_1 & f_0 & \cdots & f_2 \\ \vdots & \vdots & & \vdots \\ f_{p-1} & f_{p-2} & \cdots & f_0 \end{pmatrix}}_{T_F} \underbrace{\begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{p-1} \end{pmatrix}}_{\vec{g}}$$

where f_i (resp. g_i, h_i) is the coefficient of degree i of F (resp. G, H). Computing $H(\alpha)$ corresponds to the inner product $\vec{\alpha}_p \vec{h} = \vec{\alpha}_p T_F \vec{g}$ where $\vec{\alpha}_p = (1, \alpha, \ldots, \alpha^{p-1})$. This evaluation can be computed in $\mathcal{O}(p)$ operations in R [6]. Here we reuse similar techniques in the context of sparse polynomials.

To compute $H(\alpha)$, we first compute $\vec{c} = \vec{\alpha}_p T_F$, and then the inner product $\vec{c}\vec{g}$. If $\mathrm{supp}(G) = \{j_1, \ldots, j_{\#G}\}$ with $j_1 < \cdots < j_{\#G} < p$, we only need the corresponding entries of $\vec{c}$, that is all c_{j_k}'s for $1 \leq k \leq \#G$. Since $c_j = \sum_{\ell=0}^{p-1} \alpha^\ell f_{(\ell-j) \bmod p}$, we can write $c_j = f_{p-j} + \alpha \sum_{\ell=0}^{p-2} \alpha^\ell f_{(\ell-j+1) \bmod p}$, that is $c_j = \alpha c_{j-1} + (1 - \alpha^p) f_{p-j}$.

Applying this relation as many times as necessary, we obtain a relation to compute $c_{j_{k+1}}$ from c_{j_k}:

$$c_{j_{k+1}} = \alpha^{j_{k+1} - j_k} c_{j_k} + (1 - \alpha^p) \sum_{\ell=j_k+1}^{j_{k+1}} \alpha^\ell f_{p-\ell}.$$

Each nonzero coefficient f_t of F appears in the definition of $c_{j_{k+1}}$ if and only if $p - j_{k+1} \leq t < p - j_k$. Thus, each f_t is used exactly once to compute all the c_{j_k}'s. Since for each summand, one needs to compute α^ℓ for some $\ell < p$, the total cost for computing all the sums is $\mathcal{O}(\#F \log p)$ operations in R. Similarly, the computation of $\alpha^{j_{k+1} - j_k} c_{j_k}$ for all k costs $\mathcal{O}(\#G \log p)$. The last remaining step is the final inner product which costs $\mathcal{O}(\#G)$ operations in R, whence the result. □

The complexity is improved to $\mathcal{O}(\log p + (\#F + \#G)\log p / \log\log p)$ using again Yao's algorithm [25] for simultaneous exponentiation.

3.2 A quasi-linear time algorithm

Given three sparse polynomials F, G and H in $R[X]$, we want to assert that $H = FG$. Our approach is to take a random prime p and to verify this assertion modulo $X^p - 1$ through modular product evaluation. This method is explicitly described in the algorithm VERIFYSP that works over any large enough integral domain R. We further extend the description and the analysis of this algorithm for the specific cases $R = \mathbb{Z}$ and $R = \mathbb{F}_q$ in the next sections.

Algorithm 1 VERIFYSP

Input: $H, F, G \in R[X]$; $0 < \epsilon < 1$.
Output: True if $FG = H$, False with probability $\geq 1 - \epsilon$ otherwise.

1: Define $c_1 > \frac{10}{3}$ and $c_2 > 1$ such that $\frac{10}{3c_1} + (1 - \frac{10}{3c_1})\frac{1}{c_2} \leq \epsilon$
2: $D \leftarrow \deg(H)$
3: **if** $\#H > \#F\#G$ or $D \neq \deg(F) + \deg(G)$ **then return** False
4: $\lambda \leftarrow \max(21, c_1(\#F\#G + \#H)\ln D)$
5: $p \leftarrow$ RANDOMPRIME$(\lambda, \frac{5}{3c_1})$
6: $(F_p, G_p, H_p) \leftarrow (F \bmod X^p - 1,\ G \bmod X^p - 1,\ H \bmod X^p - 1)$
7: Define $\mathscr{E} \subset R$ of size $> c_2 p$ and choose $\alpha \in \mathscr{E}$ randomly.
8: $\beta \leftarrow [(F_p G_p) \bmod X^p - 1](\alpha)$ ▷ using Lemma 3.1
9: **return** $\beta = H_p(\alpha)$

Theorem 3.2. *If R is an integral domain of size $\geq 2c_1c_2\#F\#G\ln D$* VERIFYSP *works as specified and it requires $\mathscr{O}_\epsilon(T\log(T\log D))$ operations in R plus $\mathscr{O}_\epsilon(T\mathsf{I}(\log D))$ bit operations where $D = \deg(H)$ and $T = \max(\#F, \#G, \#H)$.*

Proof. Step 3 dismisses two trivial mistakes and ensures that D is a bound on the degree of each polynomial.

If $FG = H$, the algorithm returns True for any choice of p and α. Otherwise, there are two sources of failure. Either $X^p - 1$ divides $FG - H$, whence $(FG)_p(\alpha) = H_p(\alpha)$ for any α. Or α is a root of the nonzero polynomial $(FG - H) \bmod X^p - 1$. Since $FG - H$ has at most $\#F\#G + \#H$ terms, the first failure occurs with probability at most $\frac{10}{3c_1}$ by Prop. 2.4. And since $(FG - H) \bmod X^p - 1$ has degree at most $p - 1$ and $\mathscr{E}$ has $c_2 p$ points, the second failure occurs with probability at most $\frac{1}{c_2}$. Altogether, the failure probability is at most $\frac{10}{3c_1} + (1 - \frac{10}{3c_1})\frac{1}{c_2}$.

Let us remark that $c_1, c_2 = \mathscr{O}(\frac{1}{\epsilon})$ and $p = \mathscr{O}(\frac{1}{\epsilon}T^2\log D)$. Step 5 requires only $\tilde{\mathscr{O}}(\log^7(\frac{1}{\epsilon}T\log D))$ bit operations by Proposition 2.2. The operations in Step 6 are T divisions by p on integers bounded by D which cost $\mathscr{O}_\epsilon(T\mathsf{I}(\log D))$ bit operations, plus T additions in R. The evaluation of $F_p G_p \bmod X^p - 1$ on α at Step 8 requires $\mathscr{O}(T\log(\frac{1}{\epsilon}T\log D))$ operations in R by Lemma 3.1. The evaluation of H_p on α costs $\mathscr{O}(T\log(T\log D))$ operations in R. Other steps have negligible cost. □

3.3 Analysis over finite fields

The first easy case is the case of large finite fields: If there are enough points for the evaluation, the generic algorithm has the same guarantee of success and a quasi-linear time complexity.

Corollary 3.3. *Let F, G and H be three polynomials of degree at most D and sparsity at most T in $\mathbb{F}_q[X]$ where $q > 2c_1c_2\#F\#G\ln(D)$. Then Algorithm* VERIFYSP *has bit complexity $\mathscr{O}_\epsilon(n\log^2(n)4^{\log^* n})$ where $n = T(\log D + \log q)$ is the input size.*

Proof. By definition of n, the cost of step 6 is $\mathscr{O}_\epsilon(n\log n)$ bit operations. Each ring operation in $\mathbb{F}_q$ costs $\mathscr{O}(\log(q)\log\log(q)4^{\log^* q})$ bit operations which implies that the bit complexity of step 8 is $\mathscr{O}_\epsilon(T\log(T\log D)\log(q)\log\log(q)4^{\log^* q})$. Since $T\log q$ and $T\log D$ are bounded by n, $\log\log q \leq \log n$, the result follows. □

We shall note that even if $q < 2c_1c_2\#F\#G\ln(D)$ we can make our algorithm work by using an extension field and this approach achieves the same complexity.

Theorem 3.4. *One can adapt algorithm* VERIFYSP *to work over finite fields $\mathbb{F}_q$ such that $q < 2c_1c_2\#F\#G\ln(D)$. The bit complexity is $\mathscr{O}_\epsilon(n\log(n)\log\log(n)\,4^{\log^* n})$, where $n = T(\log D + \log q)$ is the input size.*

Proof. To have enough elements in the set $\mathscr{E}$, we need to work over $\mathbb{F}_{q^s}$ where $q^s > c_2 p \geq q^{s-1}$. An irreducible degree-s polynomial can be computed in $\tilde{\mathscr{O}}(s^3\log q) = \tilde{\mathscr{O}}(\log(T\log D)/\log q)$ by Proposition 2.6. Since α is taken in $\mathbb{F}_{q^s}$, the complexity becomes $\mathscr{O}_\epsilon(T\mathsf{I}(\log D) + T\log(T\log D)\mathsf{M}_q(s))$ bit operations. Remarking that $T \leq D$ we have $T\log(T\log D) \leq T\log(D\log D) = \mathscr{O}(n)$. Since $s\log q = \mathscr{O}(\log(T\log D)) = \mathscr{O}(\log n)$ we can obtain $\mathsf{M}_q(s) = \mathscr{O}(\log(n)\log\log(n)4^{\log^* n})$ which implies that the second term of the complexity is $\mathscr{O}(n\log(n)\log\log(n)4^{\log^* n})$. The first term is negligible since it is $\mathscr{O}(n\log n)$.

In order to achieve the same probability of success, we fix an error probability $1/c_3 < 1$ for Proposition 2.6 and we take constants c_1 and c_2 in VERIFYSP such that $1 - (1 - \frac{10}{3c_1})(1 - \frac{1}{c_2})(1 - \frac{1}{c_3}) \leq \epsilon$. □

We note that for very sparse polynomials over some fields, the complexity is only dominated by the operations on the exponents.

Corollary 3.5. VERIFYSP *has bit complexity* $\mathcal{O}_\epsilon(n \log n)$ *in the following cases :*

(i) $s = 1$ *and* $\log q = \mathcal{O}(\log^{1-\alpha} D)$ *for some constant* $0 < \alpha < 1$,

(ii) $s > 1$ *and* $T = \Theta(\log^k D)$ *for some constant* k.

Proof. In both cases the cost of reducing the exponents *modulo* p is $\mathcal{O}_\epsilon(n \log n)$ bit operations. In the first case, each multiplication in $\mathbb{F}_q$ costs $\mathcal{O}(\log(q) \log\log(q) 4^{\log^* q}) = \mathcal{O}(\log D)$ bit operations as $\log\log(q) 4^{\log^* q} = \mathcal{O}(\log^\alpha D)$. In the second case, $n = \mathcal{O}(\log^{k+1} D)$ and $s \log q = \mathcal{O}_\epsilon(\log(T^2 \log D)) = \mathcal{O}_\epsilon(\log\log D)$ which implies $\mathsf{M}_q(s) = \mathcal{O}_\epsilon(s \log q \log(s \log q) 4^{\log^* s}) = \mathcal{O}_\epsilon(\log D)$. In both cases, the algorithm performs $\mathcal{O}_\epsilon(T \log T \log D) = \mathcal{O}_\epsilon(T \log n)$ operations in $\mathbb{F}_q$ (or in $\mathbb{F}_{q^s}$). Therefore the bit complexity is $\mathcal{O}_\epsilon(n \log n)$. □

The following generalization will be used in our quasi-linear multiplication algorithm given in Section 4.

Corollary 3.6. *Let* $(F_i, G_i)_{0 \le i < m}$ *and* H *be sparse polynomials over* $\mathbb{F}_q$ *of degree at most* D *and sparsity at most* T. *We can verify if* $\sum_{i=0}^{m-1} F_i G_i = H$, *with error probability at most* ϵ *when they are different, in* $\mathcal{O}_\epsilon(m(T \mathsf{I}(\log D) + T \log(mT \log D) \mathsf{M}_q(s)))$ *bit operations.*

3.4 Analysis over the integers

In order to keep a quasi-linear time complexity over the integers, we must work over a prime finite field $\mathbb{F}_q$ to avoid the computation of too large integers. Indeed, $H_p(\alpha)$ could have a size of $p \log \alpha = \mathcal{O}_\epsilon(T^2 \log D \log \alpha)$ which is not quasi-linear in the input size.

Theorem 3.7. *One can adapt algorithm* VERIFYSP *to work over the integers. The bit complexity is* $\mathcal{O}_\epsilon(n \log n \log\log n)$, *where* $n = T(\log D + \log C)$ *is the input size with* $C = \max(\|F\|_\infty, \|G\|_\infty, \|H\|_\infty)$.

Proof. Before step 6, we choose a random prime number $q = \text{RANDOMPRIME}(\mu, \frac{5}{3c_2})$ with $\mu = c_2 \max(p, \ln(C^2 T + C))$ and we perform all the remaining step modulo q. Let us assume that the polynomial $\Delta = FG - H \in \mathbb{Z}[X]$ is nonzero. Our algorithm only fails in the following three cases: p is such that $\Delta_p = \Delta \bmod X^p - 1 = 0$; q is such that $\Delta_p \equiv 0 \bmod q$; α is a root of Δ_p in $\mathbb{F}_q$.

Using Proposition 2.4, Δ_p is nonzero with probability at least $1 - \frac{10}{3c_1}$. Actually, with the same probability, the proof of the proposition shows that at least one coefficient of Δ is preserved in Δ_p. Since $\|\Delta\|_\infty \le C^2 T + C$, Proposition 2.5 ensures that $\Delta_p \not\equiv 0 \bmod q$ with probability at least $1 - \frac{10}{3c_2}$. Finally, q has been chosen so that $\mathbb{F}_q$ has at least $c_2 p$ elements whence α is not a root of $\Delta_p \bmod q$ with probability at least $1 - \frac{1}{c_2}$. Altogether, taking $c_1, c_2 \ge \frac{10}{3}$ such that $1 - (1 - \frac{10}{3c_1})(1 - \frac{10}{3c_2})(1 - \frac{1}{c_2}) \le \epsilon$, our adaptation of VERIFYSP has an error probability at most ϵ.

The reductions of F, G and H *modulo* q add a term $\mathcal{O}(T \mathsf{I}(\log C))$ to the complexity. Since operations in $\mathbb{F}_q$ have cost $\mathsf{I}(\log q)$, the complexity becomes $\mathcal{O}(T \mathsf{I}(\log D) + T \mathsf{I}(\log C) + T \log(T \log D) \mathsf{I}(\log q))$ bit operations. The first two terms are in $\mathcal{O}(n \log n)$. Moreover, $q = \mathcal{O}_\epsilon(\log(C^2 T) + p)$ and $p = \mathcal{O}_\epsilon(T^2 \log D)$, thus $\log q = \mathcal{O}_\epsilon(\log(\log C + T \log D)) = \mathcal{O}_\epsilon(\log n)$. Since $T \le D$, $T \log(T \log D) = \mathcal{O}(n)$ and the third term in the complexity is $\mathcal{O}_\epsilon(n \log n \log\log n)$. □

As over small finite fields, the complexity is actually better for very sparse polynomials.

Corollary 3.8. *If* $T = \Theta(\log^k D)$ *for some* k, VERIFYSP *has bit complexity* $\mathcal{O}_\epsilon(n \log n)$.

Proof. If $T = \Theta(\log^k D)$, $T \log(T \log D) = \tilde{\mathcal{O}}(\log^k D) = o(n)$, thus the last term of the complexity in the proof of Theorem 3.7 becomes negligible with respect to the first two terms. □

For the same reason as for finite fields, we extend the verification algorithm to a sum of product.

Corollary 3.9. *Let* $(F_i, G_i)_{0 \le i < m}$ *and* H *be sparse polynomials of degree at most* D, *sparsity at most* T, *and height at most* C. *We can verify if* $\sum_{i=0}^{m-1} F_i G_i = H$, *with probability of error at most* ϵ *when they are different, in* $\mathcal{O}_\epsilon(mT \mathsf{I}(\log D) + mT \mathsf{I}(\log C) + mT \log(mT \log D) \mathsf{I}(\log(m \log C + mT \log D)))$ *bit operations.*

We will only use this algorithm with $m = 2$ and thus refer to it as VERIFYSUMSP$(H, F_0, G_0, F_1, G_1, \epsilon)$.

4 Sparse polynomial multiplication

Given two sparse polynomial F and G, our algorithm aims at computing the product $H = FG$ through sparse polynomial interpolation. We avoid the difficulty of computing an *a priori* bound on the sparsity of H needed for sparse interpolation by using our verification algorithm of Section 3. Indeed, one can start with an arbitrary small sparsity and double it until the interpolated polynomial matches the product according to VERIFYSP.

The remaining difficulty is to be able to interpolate H in quasi-optimal time given a sparsity bound, which is not yet achieved in the general case. In our case, we first analyze the complexity of Huang's sparse interpolation algorithm [10] when the input is a sum of sparse products. In order to obtain the desired complexity we develop a novel approach that interleaves two levels of Huang's algorithm.

4.1 Analysis of Huang's sparse interpolation

In [10] Huang proposes an algorithm that interpolates a sparse polynomial H from its SLP representation, achieving the best known complexity for this problem, though it is not optimal. Its main idea is to use the dense polynomials $H_p = H \bmod X^p - 1$ and $H'_p = H' \bmod X^p - 1$ where H' is the derivative of H and p a *small* random prime. Indeed, if cX^e is a term of H that does not collide during the reduction *modulo* $X^p - 1$, H_p contains the monomial $cX^{e \bmod p}$ and H'_p contains $ceX^{e-1 \bmod p}$, hence c and e can be recovered by a mere division. Of course, the choice of p is crucial for the method to work. It must be not too large to get a low complexity, but large enough for collisions to be sufficiently rare.

Lemma 4.1. *There exists an algorithm* FINDTERMS *that takes as inputs a prime p, two polynomials $H_p = H \bmod X^p - 1$, $H'_p = H' \bmod X^p - 1$, and bounds $D \geq \deg(H)$ and $C \geq \|H\|_\infty$ and it outputs an approximation H^* of H that contains at least all the monomials of H that do not collide* modulo $X^p - 1$. *Its bit complexity is $\mathcal{O}(T\mathsf{I}(\log CD))$, where $T = \#H$.*

Proof. It is a straightforward adaptation of [10, Algorithm 3.4 (UTERMS)]. Here, taking C as input allows us to only recover coefficients that are at most C in absolute value and therefore we perform divisions with integers of bitlength at most $\log CD$. □

Corollary 4.2. *Let H be a sparse polynomial such that $\#H \leq T$, $\deg H \leq D$ and $\|H\|_\infty \leq C$, and $0 < \epsilon < 1$. If $\lambda = \max(21, \frac{10}{3\epsilon} T^2 \ln D)$ and $p = \text{RANDOMPRIME}(\lambda, \frac{\epsilon}{2})$, then with probability at least $1 - \epsilon$,* FINDTERMS $(p, H \bmod X^p - 1, H' \bmod X^p - 1, D, C)$ *returns H.*

Proof. With probability at least $1-\epsilon$, no collision occurs in $H \bmod X^p - 1$, and consequently neither in $H' \bmod X^p - 1$, by Proposition 2.3. In this case FINDTERMS correctly computes H, according to Lemma 4.1. □

Theorem 4.3. *There exists an algorithm* INTERPSUMSP *that takes as inputs $2m$ sparse polynomials $(F_i, G_i)_{0 \leq i < m}$, three bounds $T \geq \#H$, $D > \deg(H)$ and $C \geq \|H\|_\infty$ where $H = \sum_{i=0}^{m-1} F_i G_i$, a constant $0 < \mu < 1$ and the list $\mathscr{P}$ of the first $2N$ primes for $N = \max(1, \lfloor \frac{32}{5}(T-1)\log D \rfloor)$, and outputs H with probability at least $1 - \mu$.*

Its bit complexity is $\tilde{\mathcal{O}}_\mu(mT_1 \log D_1 \log(C_1 D_1))$ where T_1, D_1 and C_1 are bounds on the sparsity, the degree and the height of H and each F_i and G_i.

Proof. It is identical to the proof of [10, Algorithm 3.9 (UIPOLY)] taking into account that H is not given as an SLP anymore but as $\sum_{i=0}^{m-1} F_i G_i$ where the polynomials F_i and G_i are given as sparse polynomials. □

Remark 4.4. *A finer analysis of algorithm* INTERPSUMSP *leads to a bit complexity $\mathcal{O}_\mu(m \log T_1 \mathsf{M}_\mathbb{Z}(T_1 \log D_1 \log(T_1 \log D_1), T_1 C_1 D_1)$.*

Remark 4.5. *Even when* INTERPSUMSP *returns an incorrect polynomial, it has sparsity at most $2T$, degree less than D and coefficients bounded by C.*

4.2 Multiplication

Our idea is to compute different candidates to FG with a growing sparsity bound and to verify the result with VERIFYSP. Unfortunately, a direct call to INTERPSUMSP with the correct sparsity $T = \max(\#F, \#G, \#(FG))$ yields a bit complexity $\tilde{\mathcal{O}}(T \log D \log CD)$ when the coefficients are bounded by C and the degree by D. We shall remark that it is not nearly optimal since the input and output size are bounded by $T \log D + T \log C$.

To circumvent this difficulty, we first compute the reductions $F_p = F \bmod X^p - 1$ and $G_p = G \bmod X^p - 1$ of the input polynomials, as well as the reductions $F'_p = F' \bmod X^p - 1$ and $G'_p = G' \bmod X^p - 1$ of their

derivatives, for a random prime p as in Corollary 4.2. The polynomials $H_p = FG \bmod X^p - 1$ and $H'_p = (FG)' \bmod X^p - 1$ can be computed using INTERPSUMSP and VERIFYSP. Indeed, we first compute F_pG_p by interpolation and then reduce it *modulo* $X^p - 1$ to get H_p. Similarly for H'_p we first interpolate $F'_pG_p + F_pG'_p$ before its reduction. Finally we can compute the polynomial FG from H_p and H'_p using FINDTERMS according to Corollary 4.2. Our choice of p, which is polynomial in the input size, ensures that each call to INTERPSUMSP remains quasi-linear.

Algorithm 2 SPARSEPRODUCT

Input: $F, G \in \mathbb{Z}[X]$. $0 < \mu_1, \mu_2 < 1$ with $\frac{\mu_1}{2} \leq \mu_2$.
Output: $H \in \mathbb{Z}[X]$ s.t. $H = FG$ with probability at least $1 - \mu_1$.

1: $t \leftarrow \max(\#F, \#G)$, $D \leftarrow \deg(F) + \deg(G)$, $C \leftarrow t\|F\|_\infty\|G\|_\infty$
2: $\lambda \leftarrow \max(21, \frac{20}{3\mu_1}(\#F\#G)^2 \ln D)$, $\mu^* \leftarrow \mu_2 - \frac{\mu_1}{2}$
3: $p \leftarrow$ RANDOMPRIME$(\lambda, \frac{\mu_1}{4})$
4: $F_p \leftarrow F \bmod X^p - 1$, $G_p \leftarrow G \bmod X^p - 1$
5: $F'_p \leftarrow F' \bmod X^p - 1$, $G'_p \leftarrow G' \bmod X^p - 1$
6: **repeat**
7: $\quad N \leftarrow \max(1, \lfloor \frac{32}{5}(t-1)\log D \rfloor)$
8: $\quad \mathscr{P} \leftarrow$ {the first $2N$ primes in increasing order}
9: $\quad H_1 \leftarrow$ INTERPSUMSP$([(F_p, G_p)], t, 2p, C, \frac{\mu^*}{2}, \mathscr{P})$
10: $\quad H_2 \leftarrow$ INTERPSUMSP$([(F_p, G'_p), (F'_p, G_p)], t, 2p, CD, \frac{\mu^*}{2}, \mathscr{P})$
11: $\quad t \leftarrow 2t$
12: **until**
$\quad$ VERIFYSP$(H_1, F_p, G_p, \frac{\mu_1}{2})$ **and** $\triangleright\ H_1 = F_pG_p$
$\quad$ VERIFYSUMSP$(H_2, F_p, G'_p, F'_p, G_p, \frac{\mu_1}{2})$ $\triangleright\ H_2 = F'_pG_p + F_pG'_p$
13: $H_p \leftarrow H_1 \bmod X^p - 1$, $H'_p \leftarrow H_2 \bmod X^p - 1$.
14: **return** FINDTERMS(p, H_p, H'_p, D, C).

Lemmas 4.6 and 4.7 respectively provide the correctness and complexity bound of algorithm SPARSEPRODUCT. Together, they consequently form a proof of Theorem 1.1 by taking $\epsilon = \mu_1 + \mu_2$. Note that this approach translates *mutatis mutandis* to the multiplication of sparse polynomials over $\mathbb{F}_q$ where the characteristic of $\mathbb{F}_q$ is larger than D.

Lemma 4.6. *Let F and G be two sparse polynomials over $\mathbb{Z}$. Then algorithm* SPARSEPRODUCT *returns FG with probability at least $1 - \mu_1$.*

Proof. Since FG has sparsity at most $\#F\#G$, Corollary 4.2 implies that if $H_p = FG \bmod X^p - 1$ and $H'_p = (FG)' \bmod X^p - 1$, the probability that FINDTERMS does not return FG is at most $\frac{\mu_1}{2}$. The other reason for the result to be incorrect is that one of these equalities does not hold, which means that one of the two verifications fails. Since this happens with probability at most $\frac{\mu_1}{2}$, SPARSEPRODUCT returns FG with probability at least $1 - \mu_1$. □

Lemma 4.7. *Let F and G be two sparse polynomials over $\mathbb{Z}$, $T = \max(\#F, \#G, \#(FG))$, $D = \deg(FG)$, $C = \max(\|F\|_\infty, \|G\|_\infty, \|FG\|_\infty)$ and $\epsilon = \mu_1 + \mu_2$. Then algorithm* SPARSEPRODUCT *has bit complexity $\tilde{\mathcal{O}}_\epsilon(T(\log D + \log C))$ with probability at least $1 - \mu_2$. Writing $n = T(\log D + \log C)$, the bit complexity is $\mathcal{O}_\epsilon(n \log^2 n \log^2 T(\log T + \log\log n))$.*

Proof. In order to obtain the given complexity, we first need to prove that with high probability INTERPSUMSP never computes polynomials with a sparsity larger than $4\#(FG)$.

Let $T_p = \max(\#(F_pG_p), \#(F_pG'_p + F'_pG_p))$. If $t \leq 2T_p$ then the polynomials H_1 and H_2 satisfy $\#H_1, \#H_2 \leq 4T_p$ by Remark 4.5. Unfortunately, T_p could be as large as T^2 and t might reach values larger than T_p. We now prove that: *(i)* with probability at least $1 - \mu^*$ the maximal value of t during the algorithm is less than $2T_p$; *(ii)* with probability at least $1 - \frac{\mu_1}{2}$, $T_p \leq \#(FG)$. Together, this will prove that $\#H_1, \#H_2 \leq 4\#(FG)$ with probability at least $1 - \mu^* - \frac{\mu_1}{2} = 1 - \mu_2$.

(i) As soon as $t \geq T_p$, Steps 9 and 10 compute both F_pG_p and $F_pG'_p + F'_pG_p$ with probability at least $1 - \mu^*$ by Theorem 4.3. Since VERIFYSP never fails when the product is correct, the algorithm ends when $T_p \leq t < 2T_p$ with probability at least $1 - \mu^*$.

(ii) Let us define the polynomials $\hat{F}_p$ and $\hat{G}_p$ obtained from F_p and G_p by replacing each nonzero coefficient by 1. The choice of p in Step 3 ensures that with probability at least $1 - \frac{\mu_1}{2}$ there is no collision in $(\hat{F}_p\hat{G}_p) \bmod$

X^p-1 by applying Proposition 2.3 to the product $\hat{F}_p\hat{G}_p$. In that case, there is also no collision in $F_pG_p \bmod X^p-1$ and in $F_pG_p' + F_p'G_p \bmod X^p-1$ since $\mathrm{supp}(F_pG_p) \subset \mathrm{supp}(\hat{F}_p\hat{G}_p)$. Therefore, there are as many nonzero coefficients in F_pG_p as in $F_pG_p \bmod X^p-1$, which is equal to $FG \bmod X^p-1$. Thus with probability at least $1-\frac{\mu_1}{2}$ we have $\#(F_pG_p) = \#(FG) \le T$ and similarly $\#(F_p'G_p + F_pG_p') = \#((FG)') \le T$.

In the rest of the proof, we assume that the loop stops with $t \le 2T_p$ and that $T_p \le T$. In particular, the number of iterations of the loop is $\mathcal{O}(\log T)$. Since $2p = \mathcal{O}(\frac{1}{\epsilon}T^4 \log D)$, Steps 9 and 10 have a bit complexity $\tilde{\mathcal{O}}_\epsilon(T\log p \log(pCD)) = \tilde{\mathcal{O}}_\epsilon(T\log CD)$ by Theorem 4.3. Using Remark 4.5, VERIFYSP and VERIFYSUMSP have polynomials of height at most tCD as inputs. By Corollary 3.9, Step 12 has bit complexity $\mathcal{O}_\epsilon(T\log(T\log p)\mathsf{I}(\log CD)) = \tilde{\mathcal{O}}_\epsilon(T\log CD)$. The list $\mathscr{P}$ can be computed incrementally, adding new primes when necessary. At the end of the loop, $\mathscr{P}$ contains $\mathcal{O}(T\log 2p)$ primes, which means that it is computed in $\mathcal{O}_\epsilon(T\log p\log^2(T\log p)\log\log(T\log p))$ bit operations [24, Chapter 18], that is $\tilde{\mathcal{O}}_\epsilon(T\log\log D)$ since $\log p = \mathcal{O}(\log(T\log D))$.

The total cost for the $\mathcal{O}(\log T)$ iterations of the loop is still $\tilde{\mathcal{O}}_\epsilon(T\log(CD))$. Step 14 runs in time $\mathcal{O}_\epsilon(T\mathsf{I}(\log CD))$ by Lemma 4.1 as the coefficients of H_p' are bounded by $2TC^2D$ with $T \le D$ and $\#H_p, \#H_p' \le \#H$. Since other steps have negligible costs this yields a complexity of $\tilde{\mathcal{O}}_\epsilon(T(\log C + \log D))$ with probability at least $1-\mu_2$.

Using Remark 4.4, we can provide a more precise complexity for Steps 9 and 10 which is $\mathcal{O}_\epsilon(\log T\mathsf{M}_{\mathbb{Z}}(T\log p\log(T\log p), pDTC))$ bit operations. It is easy to observe that the $\log T$ repetitions of these steps provide the dominant term in the complexity. A careful simplification yields a bit complexity $\mathcal{O}_\epsilon(n\log^2 n\log^2 T(\log T + \log\log n)$ for SPARSEPRODUCT where $n = T(\log D + \log C)$ bounds both input and output sizes.

□

4.3 Multivariate case

Using classical Kronecker substitution [24, Chapter 8] one can extend straightforwardly SPARSEPRODUCT to multivariate polynomials. Let $F, G \in \mathbb{Z}[X_1,\ldots,X_n]$ with $\|F\|_\infty, \|G\|_\infty \le C$ and $\deg_{X_i}(F) + \deg_{X_i}(G) < d$. Writing $F_u(X) = F(X, X^d, \ldots, X^{d^{n-1}})$ and $G_u(X) = G(X, X^d, \ldots, X^{d^{n-1}})$, one can easily retrieve FG from the univariate product F_uG_u. It is easy to remark that the Kronecker substitution preserve the sparsity and the height, and it increases the degree to $\deg F_u, \deg G_u < d^n$. Considering that F and G are sparse polynomial with at most T nonzero terms, their sizes are at most $T(n\log d + \log C)$ which is exactly the sizes of F_u and G_u. Since the Kronecker and inverse Kronecker substitutions cost $\tilde{\mathcal{O}}(Tn\log d)$ bit operations, one can compute F_uG_u using SPARSEPRODUCT within the following bit complexity.

Corollary 4.8. *There exists an algorithm that takes as inputs $F, G \in \mathbb{Z}[X_1,\ldots,X_n]$ and $0 < \epsilon < 1$, and computes FG with probability at least $1-\epsilon$, using $\tilde{\mathcal{O}}_\epsilon(T(n\log d + \log C))$ bit operations where $T = \max(\#F, \#G, \#(FG))$, $d = \max_i(\deg_{X_i} FG)$ and $C = \max(\|F\|_\infty, \|G\|_\infty)$.*

Over a finite field $\mathbb{F}_{q^s}$ for some prime q, the previous technique requires that $q > d^n$ since SPARSEPRODUCT requires q to be larger than the degree. The *randomized Kronecker substitution* method introduced by Arnold and Roche [3] allows to apply SPARSEPRODUCT to fields of smaller characteristic. The idea is to define univariate polynomials $F_s(X) = F(X^{s_1},\ldots,X^{s_n})$ and $G_s(X) = G(X^{s_1},\ldots,X^{s_n})$ for some random vector $\vec{s} = (s_1,\ldots,s_n)$ such that these polynomials have much smaller degrees than those obtained with classical Kronecker substitution. As a result, we obtain an algorithm that works for much smaller q of order $\tilde{\mathcal{O}}(nd\#F\#G)$.

Our approach is to first use some randomized Kronecker substitutions to estimate the sparsity of FG by computing the sparsity of $H_s = F_sG_s$ for several distinct random vectors $\vec{s}$. With high probability, the maximal sparsity is close to the one of FG. Then, we use this information to provide a bound to some (multivariate) sparse interpolation algorithm. Note that our approach is inspired from [11] that slightly improves randomized Kronecker substitution.

Lemma 4.9. *Let $H \in \mathbb{F}_{q^s}[X_1,\ldots,X_n]$ of sparsity T, and $\vec{s}$ be a vector chosen uniformly at random in S^n where $S \subset \mathbb{N}$ is finite. The expected sparsity of $H_s(X) = H(X^{s_1},\ldots,X^{s_n})$ is at least $T(1-\frac{T-1}{\#S})$.*

Proof. If we fix two distinct exponent vectors $\vec{e}_u$ and $\vec{e}_v$ of H, they *collide* in H_s if and only if $\vec{e}_u\cdot\vec{s} = \vec{e}_v\cdot\vec{s}$. Since $\vec{e}_u \ne \vec{e}_v$, they differ at least on one component, say $e_{u,j_0} \ne e_{v,j_0}$. The equality $\vec{e}_u\cdot\vec{s} = \vec{e}_v\cdot\vec{s}$ is then equivalent to

$$s_{j_0} = \sum_{j\ne j_0} \frac{e_{v,j}-e_{u,j}}{e_{u,j_0}-e_{v,j_0}} s_j.$$

Writing Y for the right-hand side of this equation we have

$$\Pr[\vec{e}_u \cdot \vec{s} = \vec{e}_v \cdot \vec{s}] = \Pr[s_{j_0} = Y] = \sum_y \Pr[s_{j_0} = Y | Y = y] \Pr[Y = y]$$

where the (finite) sum ranges over all possible values y of Y. Since s_{j_0} is chosen uniformly at random in S, $\Pr[s_{j_0} = Y | Y = y] = \Pr[s_{j_0} = y] \leq 1/\#S$ and the probability that $\vec{e}_u$ and $\vec{e}_v$ collide is at most $1/\#S$. This implies that the expected number of vectors that collide is at most $T(T-1)/\#S$. □

Corollary 4.10. *Let H be as in Lemma 4.9 and $\vec{v}_1, \ldots, \vec{v}_\ell \in S^n$ be some vectors chosen uniformly and independently at random. Then* $\Pr[\max_i \#H_{v_i} \leq T(1 - 2\frac{T-1}{\#S})] \leq 1/2^\ell$.

Proof. For each $\vec{v}_i$, the expected number of terms that collide in $H_{v_i}(X)$ is at most $T(T-1)/\#S$ by Lemma 4.9. Using Markov's inequality, we have $\Pr[\#H_{v_i} \leq T - 2T(T-1)/\#S] \leq 1/2$. Since the vectors $\vec{v}_i$ are independent, the result follows. □

Algorithm 3 SPARSITYESTIMATE

Input: $F, G \in \mathbb{F}_{q^s}[X_1, \ldots, X_n]$, $0 < \epsilon < 1$, $\lambda > 1$.
Output: An integer t such that $t \leq \lambda \#(FG)$.

1: $N \leftarrow \lceil 2\frac{\#F\#G-1}{1-1/\lambda} \rceil$, $\ell \leftarrow \lceil \log \frac{2}{\epsilon} \rceil$.
2: $t' \leftarrow 0$, $\mu \leftarrow \frac{\epsilon}{4\ell}$.
3: **repeat** ℓ times
4: $\quad \vec{s} \leftarrow$ random element of $\{0, \ldots, N-1\}^n$.
5: $\quad F_s \leftarrow F(X^{s_1}, \ldots, X^{s_n})$, $G_s \leftarrow G(X^{s_1}, \ldots, X^{s_n})$
6: $\quad H_s \leftarrow$ SPARSEPRODUCT(F_s, G_s, μ, μ)
7: $\quad t' \leftarrow \max(t', \#H_s)$
8: **return** $\lambda t'$.

Lemma 4.11. *Algorithm* SPARSITYESTIMATE *is correct when $q \geq \frac{4D\#F\#G}{1-1/\lambda}$ where $D = \max(\deg F, \deg G)$. With probability at least $1 - \epsilon$, it returns an integer $t \geq \#(FG)$ using $\tilde{\mathcal{O}}_\epsilon(T(n \log d + s \log q))$ bit operations where $T = \max(\#(FG), \#F, \#G)$ and $d = \max_i(\deg_{X_i} FG)$.*

Proof. Since each polynomial H_s has sparsity at most $\#(FG)$, SPARSITYESTIMATE returns an integer bounded by $\lambda\#(FG)$. SPARSEPRODUCT can be used in step 5 since $\deg H_s = \deg F_s + \deg G_s \leq 2ND \leq q$ by the definition of N. Assuming that SPARSEPRODUCT returns no incorrect answer during the loop, Corollary 4.10 applied to the product FG implies that $t' \geq \#(FG)(1 - 2(\#(FG)-1)/N)$ with probability $\geq 1 - \epsilon/2$ at the end of the loop. By definition of N and since $\#F\#G \geq \#(FG)$, $t' \geq \#(FG)/\lambda$. Taking into account the probability of failure of SPARSEPRODUCT, the probability that $\lambda t' \geq \#(FG)$ is at least $1 - \frac{3\epsilon}{4}$.

The computation of F_s and G_s requires $\mathcal{O}(Tn\mathsf{I}(\log \max(d, N)) + Ts \log q)$ bit operations in Step 5. Since $\max(\#F_s, \#G_s, \#H_s) \leq T$ and $\deg H_s = \mathcal{O}(ndT^2)$ in Step 6, the bit complexity of each call to SPARSEPRODUCT is $\tilde{\mathcal{O}}_\mu(T(\log(nd) + s \log q))$ with probability at least $1 - \mu$ using Lemma 4.7. Therefore, SPARSITYESTIMATE requires $\tilde{\mathcal{O}}_\epsilon(T(n \log d + s \log q))$ bit operations with probability at least $1 - \epsilon/4$. Together with the probability of failure this concludes the proof. □

Theorem 4.12. *There exists an algorithm that takes as inputs two sparse polynomials F and G in $\mathbb{F}_{q^s}[X_1, \ldots, X_n]$ and $0 < \epsilon < 1$ that returns the product FG in $\tilde{\mathcal{O}}_\epsilon(nT(\log d + s \log q))$ bit operations with probability at least $1 - \epsilon$, where $T = \max(\#F, \#G, \#(FG))$, $d = \max_i(\deg_{X_i} FG)$, $D = \deg FG$ and assuming that $q = \Omega(D\#F\#G + DT \log D \log(T \log D))$.*

Proof. The algorithm computes an estimate t on the sparsity of FG using SPARSITYESTIMATE$(F, G, \frac{\epsilon}{2}, \lambda)$ for some constant λ. The second step interpolates FG using Huang and Gao's algorithm [11, Algorithm 5 (MULPOLYSI)] which is parameterized by a univariate sparse interpolation algorithm. Originally, its inputs are a polynomial given as a blackbox and bounds on its degree and sparsity. In our case, the blackbox is replaced by F and G, the sparsity bound is t and the univariate interpolation algorithm is SPARSEPRODUCT.

The algorithm MULPOLYSI requires $\mathcal{O}_\epsilon(n \log t + \log^2 t)$ interpolation of univariate polynomials with degree $\tilde{\mathcal{O}}(tD)$ and sparsity at most t. Each interpolation with SPARSEPRODUCT is done with μ_1, μ_2 such that $\mu_1 + \mu_2 = \epsilon/4(n+1)\log t$, so that MULPOLYSI returns the correct answer in $\tilde{\mathcal{O}}_\epsilon(nT(\log d + s \log q))$ bit operations with probability at least $1 - \frac{\epsilon}{2}$ [11, Theorem 6]. Altogether, our two-step algorithm returns the correct answer using $\tilde{\mathcal{O}}_\epsilon(nT(\log d + s \log q))$ bit operations with probability at least $1 - \epsilon$. The value of q is such that it bounds the degrees of the univariate polynomials returned by SPARSEPRODUCT during the algorithm. □

4.4 Small characteristic

We now consider the case of sparse polynomial multiplication over a field $\mathbb{F}_{q^s}$ with characteristic smaller than the degree of the product FG (or, in the multivariate case, smaller than the degree of the product after randomized Kronecker substitution). We can no more use Huang's interpolation algorithm since it uses the derivative to encode the exponents into the coefficients and thus it only keeps the value of the exponents *modulo* q. Our idea to circumvent this problem is similar to the one in [2] that is to rather consider the polynomials over $\mathbb{Z}$ before calling our algorithm SPARSEPRODUCT.

The following proposition is only given for the multivariate case as it encompasses univariate's one. It matches exactly with the complexity result given by Arnold and Roche [2].

Proposition 4.13. *There exists an algorithm that takes as inputs two sparse polynomials F and G in $\mathbb{F}_{q^s}[X_1,\ldots,X_n]$ and $0<\epsilon<1$ that returns the product FG in $\tilde{\mathcal{O}}_\epsilon(S(n\log d + s\log q))$ bit operations with probability at least $1-\epsilon$, where S is the structural sparsity of FG and $d=\max_i(\deg_{X_i} FG)$.*

Proof. If $s=1$, the coefficients of F and G map easily to the integers in $\{0,\ldots,q-1\}$. Therefore, the product FG can be obtained by using an integer sparse polynomial multiplication, as the one in Corollary 4.8, followed by some reductions *modulo* q. Unfortunately, mapping the multiplication over the integers implies that the cancellations that could have occur in $\mathbb{F}_q$ do not hold anymore. Consequently, the support of the product in $\mathbb{Z}$ before modular reduction is exactly the structural support of FG.

If $s>1$, the coefficients of F and G are polynomials over $\mathbb{F}_q$ of degree $s-1$. As previously, mapping $\mathbb{F}_q$ to integers, F and G can be seen as $F_Y, G_Y \in \mathbb{Z}[Y][X_1,\ldots,X_n]$ where the coefficients are polynomials in $\mathbb{Z}[Y]$ of degree at most $s-1$ and height at most $q-1$.

If $T=\max(\#F,\#G)$, the coefficients of F_YG_Y are polynomials of degree at most $2s-2$ and height at most Tsq^2. Therefore, the product $FG\in\mathbb{F}_{q^s}$ can be computed by: *(i)* computing $F_B, G_B \in \mathbb{Z}[X_1,\ldots,X_n]$ by evaluating the coefficients of F_Y and G_Y at $B=Tsq^2$ (Kronecker substitution); *(ii)* computing the product $H_B=F_BG_B$; *(iii)* writing the coefficients of H_B in base B to obtain $H_Y=F_YG_Y$ (Kronecker segmentation); *(iv)* and finally mapping back the coefficients of H_Y from $\mathbb{Z}[Y]$ to $\mathbb{F}_{q^s}$.

Similarly as the case $s=1$, H_B and then H_Y have at most S nonzero coefficients. The Kronecker substitutions in *(i)* require $\tilde{\mathcal{O}}(Ts\log q)$ bit operations, while the Kronecker segmentations in *(iii)* need $\tilde{\mathcal{O}}(Ss\log q)$ bit operations. In *(iv)* we first compute Ss reductions *modulo* q on integers smaller than B, and then S polynomial divisions in $\mathbb{F}_q[Y]$ with polynomial of degree $\mathcal{O}(s)$. Thus, it can be done in $\tilde{\mathcal{O}}(Ss\log q)$ bit operations. Finally the computation in *(ii)* is dominant and it requires $\tilde{\mathcal{O}}_\epsilon(S(n\log d + s\log q))$ bit operations with probability at least $1-\epsilon$ using Corollary 4.8. □

References

[1] ARNOLD, A. *Sparse polynomial interpolation and testing*. PhD thesis, University of Waterloo, 2016.

[2] ARNOLD, A., AND ROCHE, D. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *ISSAC'15* (2015), ACM, pp. 29–36.

[3] ARNOLD, A., AND ROCHE, D. S. Multivariate sparse interpolation using randomized Kronecker substitutions. In *ISSAC'14* (2014), ACM, pp. 35–42.

[4] BEN-OR, M., AND TIWARI, P. A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. In *STOC'88* (1988), ACM, pp. 301–309.

[5] COLE, R., AND HARIHARAN, R. Verifying candidate matches in sparse and wildcard matching. In *STOC'02* (2002), ACM, pp. 592–601.

[6] GIORGI, P. A probabilistic algorithm for verifying polynomial middle product in linear time. *Information Processing Letters 139* (2018), 30–34.

[7] HARVEY, D., AND VAN DER HOEVEN, J. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *J. Complexity 54* (2019).

[8] HARVEY, D., AND VAN DER HOEVEN, J. Integer multiplication in time O(n log n). manuscript, 2019.

[9] HOEVEN, J. V. D., AND LECERF, G. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Computation 50* (2013), 227–254.

[10] HUANG, Q.-L. Sparse polynomial interpolation over fields with large or zero characteristic. In *ISSAC'19* (2019), ACM, pp. 219–226.

[11] HUANG, Q.-L., AND GAO, X.-S. Revisit Sparse Polynomial Interpolation Based on Randomized Kronecker Substitution. In *CASC'19* (2019), Springer, pp. 215–235.

[12] JOHNSON, S. C. Sparse polynomial arithmetic. *ACM SIGSAM Bulletin 8*, 3 (1974), 63–71.

[13] MONAGAN, M., AND PEARCE, R. Parallel sparse polynomial multiplication using heaps. In *ISSAC'09* (2009), ACM, p. 263.

[14] MONAGAN, M., AND PEARCE, R. Sparse polynomial division using a heap. *J. Symbolic Computation 46*, 7 (2011), 807–822.

[15] NAKOS, V. Nearly optimal sparse polynomial multiplication, 2019.

[16] PLAISTED, D. A. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theoretical Computer Science 31*, 1 (1984), 125–138.

[17] PRONY, R. 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. de l'École Polytechnique 1* (Floréal et Prairial III (1795)), 24–76.

[18] ROCHE, D. S. Chunky and equal-spaced polynomial multiplication. *J. Symbolic Computation 46*, 7 (2011), 791–806.

[19] ROCHE, D. S. What can (and can't) we do with sparse polynomials? In *ISSAC'18* (2018), ACM, pp. 25–30.

[20] ROSSER, J. B., AND SCHOENFELD, L. Approximate formulas for some functions of prime numbers. *Illinois J. Math. 6*, 1 (1962), 64–94.

[21] SHOUP, V. *A Computational Introduction to Number Theory and Algebra*, second ed. Cambridge University Press, 2008.

[22] VAN DER HOEVEN, J., LEBRETON, R., AND SCHOST, É. Structured FFT and TFT: Symmetric and Lattice Polynomials. In *ISSAC'13* (2013), ACM, pp. 355–362.

[23] VAN DER HOEVEN, J., AND LECERF, G. On the Complexity of Multivariate Blockwise Polynomial Multiplication. In *ISSAC'12* (2012), ACM, pp. 211–218.

[24] VON ZUR GATHEN, J., AND GERHARD, J. *Modern Computer Algebra*, 3rd ed. Cambridge University Press, 2013.

[25] YAO, A. C.-C. On the Evaluation of Powers. *SIAM Journal on Computing 5*, 1 (1976), 100–103.