Essentially optimal sparse polynomial multiplication

Pascal Giorgi, Bruno Grenet, Armelle Perret Du Cray

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

June 5, 2020

Abstract

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 extend our result to the multivariate case.

1 Introduction

Polynomials are one of the most basic objects in computer algebra and the study of fast polynomial operations remains a very challenging task. Polynomials can be represented using either the dense representation, that stores all the coefficients in a vector, or the more compact sparse representation, that only stores nonzero monomials. In the dense representation, we know quasi-optimal algorithms for decades. Yet, this is not the case for sparse polynomials.

In the sparse representation, a polynomial $F = \sum_{i=0}^{D} f_i X^i \in R[X]$ is expressed as a list of pairs (e_i, f_{e_i}) such that all the f_{e_i} are nonzero. We denote by $\#F$ its *sparsity*, i.e. the number of nonzero coefficients. Let F be a polynomial of degree D, and B a bound on the size of its coefficients. Then, the size of the sparse representation of F is $O(\#F(B + \log D))$ bits. It is common to use $B = 1 + \max_i(\lfloor \log_2(|f_{e_i}|) \rfloor)$ if $R = \mathbb{Z}$ and $B = 1 + \lfloor \log_2 q \rfloor$ if $R = \mathbb{F}_q$. The sparse representation naturally extends to polynomials in n variables: Each exponent is replaced by a vector of exponents which gives a total size of $O(\#F(B + n \log D)$.

Several problems on sparse polynomials have been investigated to design fast algorithms, including arithmetic operations, interpolation and factorization. We refer the interested readers to the excellent survey by Roche and the references therein [22]. Contrary to the dense case, note that *fast* algorithms for sparse polynomials have a (poly-)logarithmic dependency on the degree. Unfortunately, as shown by several NP-hardness results, such fast algorithms might not even exist unless $\mathsf{P} = \mathsf{NP}$. This is for instance the case for GCD computations [19].

In this paper, we are interested in the problem of sparse polynomial multiplication. In particular, we provide the first quasi-optimal algorithm whose complexity is quasi-linear in both the input and the output sizes.

1.1 Previous work

The main difficulty and the most interesting aspect of 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. 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.

The product of two polynomials of sparsity T can be computed by generating the T^2 possible monomials, sorting them by increasing degree and merging those with the same degree. Using radix sort, this algorithm takes $O(T^2(\mathsf{M}_R + \log D))$ 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 factor, even if the result has less than T^2 terms. Many improvements have been proposed to reduce this space complexity, to extend the approach to multivariate

polynomials, and to provide fast implementations in practice [15, 16, 17]. Yet, none of these results reduces the T^2 factor in the time complexity.

In general, no complexity improvement is expected as the output polynomial may have as many as T^2 nonzero coefficients. However, this number of nonzero coefficients can be overestimated, giving the opportunity for output-sensitive algorithms. Such algorithms have first been proposed for special cases. Notably, when the output size is known to be small due to sufficiently structured inputs [21], especially in the multivariate case [11, 10], or when the support of the output is known in advance [12]. It is possible to go one step further by studying the conditions for small outputs. A first reason is exponent *collisions*. Let $F = \sum_{i=1}^{T} f_i X^{\alpha_i}$ and $G = \sum_{j=1}^{T} g_j X^{\beta_j}$. A collision occurs when there exist 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 reason is coefficient cancellations. In the previous example, the resulting coefficient is $(f_{i_1} g_{j_1} + f_{i_2} g_{j_2})$, which could vanish depending on the coefficient values. Taking into account the exponent collisions amounts to computing the *sumset* of the exponents of F and G, that is $\{\alpha_i + \beta_j : 1 \leq i, j \leq T\}$. Arnold and Roche call this set the *structural support* of the product FG and its size the *structural sparsity* [3]. If $H = FG$, then the structural sparsity S of the product FG satisfies $2 \leq \#H \leq S \leq T^2$. Observe that although $\#H$ and S can be close, their difference can reach $O(T^2)$ as shown by the next example.

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

For polynomials with nonnegative integer coefficients, the support of H is exactly the sumset of the exponents of F and G, the structural support of $H = FG$. In this case, Cole and Hariharan describe a multiplication algorithm requiring $\tilde{O}(S \log^2 D)$[1] operations in the RAM model with $O(\log(CD))$ word size [5], where $\log(C)$ bounds the bitsize of the coefficients. Arnold and Roche improve this complexity to $\tilde{O}(S \log D + \#H \log C)$ bit operations for polynomials with both positive and negative integer coefficients [3]. Note that they also extend their result to finite fields and to the multivariate case. A recent algorithm of Nakos avoids the dependency on the structural sparsity for the case of integer polynomials [18], using the same word RAM model as Cole and Hariharan. Unfortunately, the bit complexity of this algorithm $(\tilde{O}((T \log D + \#H \log^2 D) \log(CD) + \log^3 D))$ is not quasi-linear.

In the dense case, quasi-optimal multiplication algorithms rely on the well-known evaluation-interpolation scheme. In the sparse settings, this approach is not efficient. The fastest multiplication algorithms mentioned above [3, 18] mainly rely on a different method called *sparse interpolation*[2], that has received considerable attention. See e.g. the early results of Prony [20] and Ben-Or and Tiwari [4] or the recent results by Huang [13]. Despite extensive analysis of this problem, no quasi-optimal algorithm exists yet. We remark that it is not the only difficulty. Simply using a quasi-optimal sparse interpolation algorithm would not be enough to get a quasi-optimal sparse multiplication algorithm [1].

1.2 Our contributions

Our main result is summarized in Theorem 1.1. We extend the complexity notations to O_ϵ and $\tilde{O}_\epsilon$ for hiding some polynomial factors in $\log(\frac{1}{\epsilon})$. Let $F = \sum_{i=1}^{T} f_i X^{e_i}$. We use $\|F\|_\infty = \max_i |f_i|$ to denote its height, $\#F$ for its number of nonzero terms and $\mathrm{supp}(F) = \{e_1, \ldots, e_T\}$ its support.

Theorem 1.1. *Given two sparse polynomials F and G over $\mathbb{Z}$, Algorithm* SparseProduct *computes $H = FG$ in $\tilde{O}_\epsilon(T(\log D + \log C))$ 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. We adapt Huang's algorithm [13] to interpolate FG in quasi-linear time. Note that the original algorithm does not reach quasi-linear complexity.

Sparse interpolation algorithms, including Huang's, require a bound on the sparsity of the result. We replaced this bound by a guess on the sparsity and an *a posteriori* verification of the product, as in [18]. However, using the classical polynomial evaluation approach for the verification does not yield a quasi-linear bit complexity (see Section 3). Therefore, we introduce a novel verification method that is essentially optimal.

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 $\tilde{O}_\epsilon(T(\log D + B))$ bit operations, where $D = \deg(H)$, B is a bound on the bitsize of the coefficients of*

[1] Here, and throughout the article, $\tilde{O}(f(n))$ denotes $O(f(n) \log^k(f(n)))$ for some constant $k > 0$.

[2] Despite their similar names, dense and sparse polynomial interpolation are actually two quite different problems.

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, using Kronecker substitution, we show that our sparse polynomial multiplication algorithm extends to the multivariate case with a quasi-linear bit complexity $\tilde{O}_\epsilon(T(n\log d + B))$ where n is the number of variables and d the maximal partial degree on each variable. Nevertheless, over finite fields this approach requires an exponentially large characteristic. Using the randomized Kronecker substitution [2] we derive a fast algorithm for finite fields of characteristic polynomial in the input size. Its bit complexity is $\tilde{O}_\epsilon(nT(\log d + B))$. Even though it is not quasi-optimal, it achieves the best known complexity for this case.

2 Preliminaries

We denote by $\mathsf{I}(n) = O(n\log n)$ the bit complexity of the multiplication of two integers of at most n bits [9]. Similarly, we denote by $\mathsf{M}_q(D) = O(D\log(q)\log(D\log q)4^{\log^* D})$ the bit complexity of the multiplication of two *dense* polynomials of degree at most D over $\mathbb{F}_q$ where q is prime [8]. The cost of multiplying two elements of $\mathbb{F}_{q^s}$ is $O(\mathsf{M}_q(s))$. The cost of multiplying two dense polynomials over $\mathbb{Z}$ of heights at most C and degrees at most D is $\mathsf{M}_{\mathbb{Z}}(D, C) = \mathsf{I}(D(\log C + \log D))$ [6, 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 (Rosser and Schoenfeld [23])**.** *If* $\lambda \geq 21$*, there are at least* $\frac{3}{5}\lambda/\ln\lambda$ *prime numbers in* $[\lambda, 2\lambda]$*.*

Proposition 2.2 ([24, Chapter 10])**.** *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* $\tilde{O}_\epsilon(\log^3 \lambda)$*.*

We need two distinct properties on 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 [3, 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. For $H \bmod X^p - 1$ to be nonzero, it is sufficient that there exists one exponent e of H that is not congruent to any other exponent 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 exist $\frac{3}{5}\lambda/\ln\lambda$ 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 next two propositions are used to reduce integer coefficients modulo some prime number and 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 ([24, 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* $\tilde{O}_\epsilon(s^3\log q)$*.*

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. (No optimal deterministic algorithm exists yet.) When $R = \mathbb{Z}$ or $\mathbb{F}_q$, a divide and conquer approach provides a quasi-linear complexity, namely $\tilde{O}(DB)$ bit operations 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 $O(T(\log D + B))$. Indeed, computing α^D requires $O(\log D)$ operations in R which implies a bit complexity of $\tilde{O}(\log(D)\log(q))$ when $R = \mathbb{F}_q$. Applying this computation to the T nonzero monomials gives a bit complexity of $\tilde{O}(T\log(D)\log(q))$. We mention that the latter approach can be improved to $\tilde{O}((1 + T/\log\log(D))\log(D)\log(q))$ using Yao's result [25] on simultaneous exponentiation. When $R = \mathbb{Z}$, the best known approach to avoid expression swell is to pick a random prime p and to perform the evaluations modulo p. One needs to choose $p > D$ in order to have a nonzero probability of success. Therefore, the bit complexity contains a $T\log^2 D$ factor.

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 $O((\#F + \#G)\log p)$ 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 $O(p)$ operations in R [7]. 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 $O(\#F \log p)$ operations in R. Similarly, the computation of $\alpha^{j_{k+1}-j_k} c_{j_k}$ for all k costs $O(\#G \log p)$. The last remaining step is the final inner product which costs $O(\#G)$ operations in R, whence the result. □

The complexity is improved to $O(\log p + (\#F + \#G)\log p/\log\log p)$ 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 $O_\epsilon(T\log(T\log D))$ operations in R plus $O_\epsilon(T\mathsf{I}(\log D))$ 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 Proposition 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 $c_1, c_2 = O(\frac{1}{\epsilon})$ and $p = O(\frac{1}{\epsilon}T^2\log D)$. Step 5 requires only $\tilde{O}(\log^3(\frac{1}{\epsilon}T\log D))$ bit operations by Proposition 2.2. The operations in Step 6 are T divisions by p on integers bounded by D which cost $O_\epsilon(T\mathsf{I}(\log D))$ bit operations, plus T additions in R. The evaluation of $F_p G_p \bmod X^p - 1$ on α at Step 8 requires $O(T\log(\frac{1}{\epsilon}T\log D))$ operations in R by Lemma 3.1. The evaluation of H_p on α costs $O(T\log(T\log D))$ operations in R. Other steps have negligible costs. □

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 $O_\epsilon(n\log^2(n)4^{\log^* n})$ where $n = T(\log D + \log q)$ is the input size.*

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

We shall note that even if $q < 2c_1c_2\#F\#G\ln(D)$ we can make our algorithm to 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 $O_\epsilon(n\log(n)\log\log(n)\,4^{\log^* n})$, 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 $\tilde{O}(s^3\log q) = \tilde{O}(\log(T\log D)/\log q)$ by Proposition 2.6. Since α is taken in $\mathbb{F}_{q^s}$, the complexity becomes $O_\epsilon(T\mathsf{I}(\log D) + T\log(T\log D)\mathsf{M}_q(s))$ bit operations. Remarking that $T \leq D$ we have $T\log(T\log D) \leq T\log(D\log D) = O(n)$. Since $s\log q = O(\log(T\log D)) = O(\log n)$ we can obtain $\mathsf{M}_q(s) = O(\log(n)\log\log(n)4^{\log^* n})$ which implies that the second term of the complexity is $O(n\log(n)\log\log(n)4^{\log^* n})$. The first term is negligible since it is $O(n\log n)$.

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* $O_\epsilon(n \log n)$ *in the following cases :*

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

The following generalization is 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* $O_\epsilon(m(T \mathsf{I}(\log D) + T \log(mT \log D) \mathsf{M}_q(s)))$ *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 size $p \log(\alpha) = O_\epsilon(T^2 \log(D) \log(\alpha))$ 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* $O_\epsilon(n \log n \log\log n)$, *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 steps 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 $O(T \mathsf{I}(\log C))$ to the complexity. Since operations in $\mathbb{F}_q$ have cost $\mathsf{I}(\log q)$, the complexity becomes $O(T \mathsf{I}(\log D) + T \mathsf{I}(\log C) + T \log(T \log D) \mathsf{I}(\log q))$ bit operations. The first two terms are in $O(n \log n)$. Moreover, $q = O_\epsilon(\log(C^2 T) + p)$ and $p = O_\epsilon(T^2 \log D)$, thus $\log q = O_\epsilon(\log(\log C + T \log D)) = O_\epsilon(\log n)$. Since $T \le D$, $T \log(T \log D) = O(n)$ and the third term in the complexity is $O_\epsilon(n \log n \log\log n)$. □

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* $O_\epsilon(n \log n)$.

Proof. If $T = \Theta(\log^k D)$, $T \log(T \log D) = \tilde{O}(\log^k D) = o(n)$, 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 products.

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* $O_\epsilon(mT \mathsf{I}(\log D) + mT \mathsf{I}(\log C) + mT \log(mT \log D) \mathsf{I}(\log(m \log C + mT \log D)))$ *bit operations.*

We shall 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 polynomials 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 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 [13] 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 [13] 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 small enough 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 $O(T\mathsf{I}(\log CD))$, where $T = \#H$.*

Proof. It is a straightforward adaptation of [13, 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 to perform divisions with integers of bitsize 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 $\tilde{O}_\mu(mT_1 \log(D_1) \log(C_1 D_1))$ 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 [13, 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 $O_\mu(m \log T_1 \mathsf{M}_\mathbb{Z}(T_1 \log(D_1) \log(T_1 \log D_1), T_1 C_1 D_1)$.*

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 $\tilde{O}(T \log(D) \log(CD))$ if 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: $\quad N \leftarrow \max(1, \lfloor \frac{32}{5}(t-1)\log p \rfloor)$
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**
VERIFYSP$(H_1, F_p, G_p, \frac{\mu_1}{2})$ **and** ▷ $H_1 = F_pG_p$
VERIFYSUMSP$(H_2, F_p, G'_p, F'_p, G_p, \frac{\mu_1}{2})$ ▷ $H_2 = F'_pG_p + F_pG'_p$
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 $\tilde{O}_\epsilon(T(\log D + \log C))$ with probability at least $1 - \mu_2$. Writing $n = T(\log D + \log C)$, the bit complexity is $O_\epsilon(n \log^2 n \log^2 T(\log T + \log\log n))$.*

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 $O(\log T)$. Since $2p = O(\frac{1}{\epsilon}T^4 \log D)$, Steps 9 and 10 have a bit complexity $\tilde{O}_\epsilon(T\log(p)\log(pCD)) = \tilde{O}_\epsilon(T\log CD)$ 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 $O_\epsilon(T\log(T\log p)\mathsf{I}(\log CD)) = \tilde{O}_\epsilon(T\log CD)$. The list $\mathscr{P}$ can be computed incrementally, adding new primes when necessary. At the end of the loop, $\mathscr{P}$ contains $O(T\log 2p)$ primes, which means that it is computed in $O_\epsilon(T\log(p)\log^2(T\log p)\log\log(T\log p))$ bit operations [6, Chapter 18], that is $\tilde{O}_\epsilon(T\log\log D)$ since $\log p = O(\log(T\log D))$.

The total cost for the $O(\log T)$ iterations of the loop is still $\tilde{O}_\epsilon(T\log(CD))$. Step 14 runs in time $O_\epsilon(T\mathsf{I}(\log CD))$ 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 $\tilde{O}_\epsilon(T(\log C + \log D))$ 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 $O_\epsilon(\log T\,\mathsf{M}_{\mathbb{Z}}(T\log(p)\log(T\log p), pDTC))$ 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 $O_\epsilon(n\log^2 n\log^2 T(\log T + \log\log n)$ for SPARSEPRODUCT where $n = T(\log D + \log C)$ bounds both input and output sizes. □

4.3 Multivariate case

Using classical Kronecker substitution [6, 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 $\deg F_u, \deg G_u < d^n$. If F and G are sparse polynomials 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 $\tilde{O}(Tn\log d)$ 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 $\tilde{O}_\epsilon(T(n\log d + \log C))$ 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 [2] 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 $\tilde{O}(nd\#F\#G)$.

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 [14] 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 $\tilde{O}_\epsilon(T(n \log d + s \log q))$ 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 $O(Tn\mathrm{l}(\log \max(d, N)) + Ts \log q)$ bit operations in Step 5. Since $\max(\#F_s, \#G_s, \#H_s) \leq T$ and $\deg H_s = O(ndT^2)$ in Step 6, the bit complexity of each call to SPARSEPRODUCT is $\tilde{O}_\mu(T(\log(nd) + s \log q))$ with probability at least $1 - \mu$ using Lemma 4.7. Therefore, SPARSITYESTIMATE requires $\tilde{O}_\epsilon(T(n \log d + s \log q))$ 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 $\tilde{O}_\epsilon(nT(\log d + s \log q))$ 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 [14, 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 $O_\epsilon(n \log t + \log^2 t)$ interpolation of univariate polynomials with degree $\tilde{O}(tD)$ 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 $\tilde{O}_\epsilon(nT(\log d + s \log q))$ bit operations with probability at least $1 - \frac{\epsilon}{2}$ [14, Theorem 6]. Altogether, our two-step algorithm returns the correct answer using $\tilde{O}_\epsilon(nT(\log d + s \log q))$ 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 [3] 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 [3].

Proposition 4.13. *There exists an algorithm that takes as inputs two sparse polynomials F and G in $\mathbb{F}_{q^s}[X_1,\dots,X_n]$ and $0<\epsilon<1$ that returns the product FG in $\tilde{O}_\epsilon(S(n\log d + s\log q))$ 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 $\{0,\dots,q-1\}$. 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 occurred 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 $F_Y, G_Y \in \mathbb{Z}[Y][X_1,\dots,X_n]$ 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 $F_Y G_Y$ 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 $F_B, G_B \in \mathbb{Z}[X_1,\dots,X_n]$ by evaluating the coefficients of F_Y and G_Y at $B=Tsq^2$ (Kronecker substitution); *(ii)* computing the product $H_B=F_B G_B$; *(iii)* writing the coefficients of H_B in base B to obtain $H_Y=F_Y G_Y$ (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 $\tilde{O}(Ts\log q)$ bit operations, while the Kronecker segmentations in *(iii)* need $\tilde{O}(Ss\log q)$ 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 $O(s)$. Thus, it can be done in $\tilde{O}(Ss\log q)$ bit operations. Finally the computation in *(ii)* is dominant and it requires $\tilde{O}_\epsilon(S(n\log d + s\log q))$ bit operations with probability at least $1-\epsilon$ using Corollary 4.8. □

References

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

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

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

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

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

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

[7] Pascal Giorgi. A probabilistic algorithm for verifying polynomial middle product in linear time. *Inform. Process. Lett.*, 139:30–34, 2018. DOI: 10.1016/j.ipl.2018.06.014.

[8] David Harvey and Joris van der Hoeven. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *J. Complexity*, 54, 2019. DOI: 10.1016/j.jco.2019.03.004.

[9] David Harvey and Joris van der Hoeven. Integer multiplication in time O(n log n), 2019. URL: https://hal.archives-ouvertes.fr/hal-02070778.

[10] Joris van der Hoeven, Romain Lebreton, and Éric Schost. Structured FFT and TFT: Symmetric and Lattice Polynomials. In *ISSAC'13*, pages 355–362. ACM, 2013. DOI: 10.1145/2465506.2465526.

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

[12] Joris van der Hoeven and Grégoire 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.

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

[14] Qiao-Long Huang and Xiao-Shan Gao. Revisit Sparse Polynomial Interpolation Based on Randomized Kronecker Substitution. In *CASC'19*, pages 215–235. Springer, 2019. DOI: 10.1007/978-3-030-26831-2_15.

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

[16] Michael Monagan and Roman Pearce. Parallel sparse polynomial multiplication using heaps. In *ISSAC'09*, page 263. ACM, 2009. DOI: 10.1145/1576702.1576739.

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

[18] Vasileios Nakos. Nearly optimal sparse polynomial multiplication, 2019. ARXIV: 1901.09355.

[19] David 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.

[20] 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.

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

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

[23] J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6(1):64–94, 1962. DOI: 10.1215/ijm/1255631807.

[24] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, second edition, 2008.

[25] Andrew Chi-Chih Yao. On the Evaluation of Powers. *SIAM J. Comput.*, 5(1):100–103, 1976. DOI: 10.1137/0205008.