Sparse Multiplication for Skew Polynomials

Mark Giesbrecht
Cheriton School of Computer Science
University of Waterloo
mwg@uwaterloo.ca

Qiao-Long Huang
Research Center for Mathematics and Interdisciplinary Sciences Shandong University
huangqiaolong@sdu.edu.cn

Éric Schost
Cheriton School of Computer Science
University of Waterloo
eschost@uwaterloo.ca

Abstract

Consider the skew polynomial ring $L[x;\sigma]$, where L is a field and σ is an automorphism of L of order r. We present two randomized algorithms for the multiplication of *sparse* skew polynomials in $L[x;\sigma]$.

The first algorithm is Las Vegas; it relies on evaluation and interpolation on a normal basis, at successive powers of a normal element. For inputs $A, B \in L[x;\sigma]$ of degrees at most d, its expected runtime is $O\tilde{}(\max(d,r)rR^{\omega-2})$ operations in K, where $K = L^\sigma$ is the fixed field of σ in L and $R \le r$ is the size of the Minkowski sum $\mathrm{supp}(A) + \mathrm{supp}(B)$ taken modulo r; here, the supports $\mathrm{supp}(A), \mathrm{supp}(B)$ are the exponents of non-zero terms in A and B.

The second algorithm is Monte Carlo; it is "super-sparse", in the sense that its expected runtime is $O\tilde{}(\log(d)Sr^\omega)$, where S is the size of $\mathrm{supp}(A) + \mathrm{supp}(B)$. Using a suitable form of Kronecker substitution, we extend this second algorithm to handle multivariate polynomials, for certain families of extensions.

Keywords

Sparse polynomials; skew polynomials; multiplication

ACM Reference Format:
Mark Giesbrecht, Qiao-Long Huang, and Éric Schost. 2020. Sparse Multiplication for Skew Polynomials. In *International Symposium on Symbolic and Algebraic Computation (ISSAC '20), July 20–23, 2020, Kalamata, Greece.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3373207.3404058

1 Introduction

Skew polynomial rings were introduced by Ore [24] as a non-commutative generalization of usual commutative polynomial rings. They have found numerous applications, as they allow one to work with linear differential equations, shift equations, or operators over finite fields, in an algebraic manner.

A very common construction is the following: let $K \subset L$ be finite fields and let $\sigma : L \to L$ be a K-automorphism of L, that is, a power of the qth power Frobenius automorphism, with $q = \#K$. For an indeterminate x over L, the ring $L[x;\sigma]$ of skew polynomials over L is the L-vector space of finite sums $A = \sum_{0 \le i \le d} a_i x^i$, with all a_i's in L, endowed with the usual addition, and where multiplication is determined by the commutation relation $xc = \sigma(c)x$ for any c in L. The *degree* $\deg(A)$ of A is the largest index i for which a_i is non-zero.

In particular, if σ is the qth power Frobenius automorphism itself, $L[x;\sigma]$ is isomorphic to the ring of linearized polynomials over K (endowed with addition and composition). Fundamental algorithms for such rings are presented in [14]. These polynomials can be used to construct algebraic codes [4, 5, 9, 27], have applications in cryptography [3, 32], underlie the construction of finite Drinfeld modules [17], etc.

In this paper, our framework is slightly more general: we assume that L is any field endowed with an automorphism σ, we let $K = L^\sigma$, and we assume that σ has finite order r; the rest of the definition is then as above. In particular, L is a separable extension of K, with $[L : K] = r$. For a list of examples that goes beyond finite fields, see Section 1 in [6].

We are interested in the cost of multiplying such skew polynomials. Given A and B in $L[x;\sigma]$ of degree at most d, the standard "schoolbook" multiplication algorithm uses $O(d^2)$ arithmetic operations $+, \times$ in L, and $O(d^2)$ applications of powers of σ. In [25, 26], Puchinger and Wachter-Zeh improved this to $O\tilde{}(d^{(\omega+1)/2})$ arithmetic operations in L and applications of powers of σ; here ω is such that over any ring, square matrix multiplication in size s can be done in $O(s^\omega)$ ring operations. The best known value to date is $\omega \le 2.373$ [7, 10], giving $(\omega + 1)/2 \le 1.69$, hence resulting in a subquadratic bound in d.

However, this analysis overlooks the (non-trivial) question of how operations in L are actually implemented. In this paper, we will measure runtimes in terms of operations in K, using the structure of L as a K-vector space; this will be our main cost measure, but we will also count bit operations when warranted (when non-trivial operations on exponents take place, for instance).

As in [6], we will use two K-bases for L. The first one, written $\mathscr{W} = (\omega_0, \ldots, \omega_{r-1})$, is taken such that addition, multiplication and inversions in L use $O\tilde{}(r)$ operations $(+, \times, \div)$ in K; here, the "soft-Oh" notation indicates that we omit polylogarithmic factors in r. For instance, if L is given as $L = K[z]/f(z)$, for some $f \in K[z]$ of degree r, then we can take ω_i to be the residue class of z^i for all i. This will be called the working basis; our convention is that *the inputs and outputs of all algorithms will be given on this basis.*

The second basis will be a *normal basis* $\mathscr{N} = (\nu_0, \ldots, \nu_{r-1})$, such that $\sigma(\nu_i) = \nu_{i+1 \bmod r}$ for all i. In such a basis, addition and application of any power of σ take linear time $O(r)$. In our algorithms, we make the following assumption about the availability of representational data for a normal basis of L/K:

ISSAC '20, July 20–23, 2020, Kalamata, Greece
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7100-1/20/07…$15.00
https://doi.org/10.1145/3373207.3404058

(H): the bases $\mathscr{W}$ and $\mathscr{N}$, as well as the matrices $\boldsymbol{M}_{\mathscr{N}\to\mathscr{W}}$ and $\boldsymbol{M}_{\mathscr{W}\to\mathscr{N}}$ of change of basis between $\mathscr{W}$ and $\mathscr{N}$, are given.

In this context, Caruso and Le Borgne [6] give a Las Vegas algorithm for multiplication in $L[x,\sigma]$ of expected cost $O\tilde{}(dr^{\omega-1})$ operations in K when $d \geq r$; for $d \leq r$, they propose another algorithm, whose cost is $O(d^{\omega-2}r^2)$ operations in K. Note that this paper also assumes that in the basis $\mathscr{W}$, the application of σ takes quasi-linear time, that is, $O\tilde{}(r)$ operations in K. This is a reasonable assumption when K and L are finite fields, as [8] [1] show that any finite field extension of a finite field admits a basis in which the operations addition, multiplication, division and application of σ cost $O\tilde{}(r)$ operations in K.

However, in our context, we show that this assumption can be dropped. This gives the advantage of more flexibility in choosing the working basis, so we will not make such an assumption.

Note that we will not address the problem of *finding* a normal basis; this has been widely studied, and we refer the reader to [11, 15, 16, 19] and references therein.

The previous discussion assumes that the input A and B are "dense polynomials", that is, given by the array of all their coefficients; in this case, in degree d, input and output size are $\Theta(dr)$ elements in K, so Caruso and Le Borgne's result of $O\tilde{}(dr^{\omega-1})$ operations in K is close to optimal (and would be optimal if we could take $\omega = 2$). In this current paper, we revisit this question, taking into account the "sparsity" of A and B. Following [2], we define the following, for a polynomial $A = \sum_{i=1}^{t} a_i x^{e_i}$, in $L[x;\sigma]$ with $0 \leq e_1 < \cdots < e_t$ and all a_i non-zero:

- the *sparsity* $\#A$ is the number t in the expression above;
- the *support* $\mathrm{supp}(A)$ is the set of exponents $\{e_1,\ldots,e_t\} \subset \mathbb{N}$.

For two polynomials A and B, we have the inequalities

$$\#(AB) \leq \#\mathbb{S}(A,B) \leq \#A \cdot \#B,$$

where $\mathbb{S}(A,B)$ is the Minkowski sum

$$\mathbb{S}(A,B) := \{e_A + e_B \mid e_A \in \mathrm{supp}(A), e_B \in \mathrm{supp}(B)\}. \tag{1.1}$$

A strict inequality $\#(AB) < \#\mathbb{S}(A,B)$ occurs only in the presence of coefficient cancellations. We will often write $S := \#\mathbb{S}(A,B)$. Recalling that r is the order of σ, we will also define

$$\mathbb{S}_r(A,B) := \{(e_A+e_B) \bmod r \mid e_A \in \mathrm{supp}(A), e_B \in \mathrm{supp}(B)\}. \tag{1.2}$$

After we discuss reduction modulo central elements, we will see that $\mathbb{S}_r(A,B)$ contains the support of the polynomial $AB \bmod (x^r-1)$. If we write $R := \#\mathbb{S}_r(A,B)$, this means that we have $\#(AB \bmod (x^r-1)) \leq R$; note also the inequalities $R \leq r$ and $R \leq S$.

In this paper, we give two randomized algorithms for multiplying skew polynomials in $L[x;\sigma]$. The first one is Las Vegas; for inputs of degree at most d, it uses an expected $O\tilde{}(\max(d,r)rR^{\omega-2})$ operations in K, where R is as above. This algorithm is based on Caruso and Le Borgne's [6]; as in that reference, the whole multiplication procedure reduces to several instances of multiplication modulo x^r-1. Whereas the original algorithm uses $O\tilde{}(r^{\omega})$ operations in K for this task, ours takes $O\tilde{}(r^2R^{\omega-2})$ operations. Altogether, since $R \leq r$, our runtime is asymptotically never worse than that in [6], and can be better in many cases. Apart from this, Puchinger and Wachter-Zeh's algorithm performs the same computation with complexity $O\tilde{}(d^{(\omega+1)/2}r)$ operations. As stated in [6], the algorithm in [25, 26] is faster than the one in [6] for polynomials of small degree $d \leq r^{2/(5-\omega)}$. As $d \leq r^{2/(5-\omega)} \leq r$, in this case, our complexity is $O\tilde{}(r^2R^{\omega-2})$. So unless $d \leq \min(r^{2/(5-\omega)}, r^{2/(\omega+1)}R^{(2\omega-4)/(\omega+1)})$, our new algorithm is faster. The precise statement is as follows.

Theorem 1.1. *Let L be a field with automorphism σ of finite order r and $K = L^{\sigma}$, and assume we have representational data* ***(H)*** *for L/K as above. Given $A,B \in L[x;\sigma]$ of degrees at most d, there is a Las Vegas algorithm to compute AB with an expected cost of $O\tilde{}(\max(d,r)rR^{\omega-2})$ operations in K and $O\tilde{}(\max(d,r))$ bit operations, where $R \leq r$ is the cardinality of the set $\mathbb{S}_r(A,B)$ defined in* (1.2).

The second algorithm comes in two stages, one of which is Monte Carlo and the other Las Vegas. Overall, for a given probability of failure, its expected runtime is now polynomial in $\log(d)$, r and S, where S is the cardinality of the set $\mathbb{S}(A,B)$ defined in (1.1). Due to this logarithmic dependence in the degree d, we will call this algorithm *supersparse.*

The algorithm is inspired by the work of Arnold and Roche [2] on the multiplication of sparse commutative polynomials. We first compute $\mathbb{S}(A,B)$; once it is known, we compute at least half the coefficients of the product AB through multiplication modulo a well-chosen central polynomial of the form $x^{pr}-1$. After a logarithmic number of iterations, this gives us the whole product AB.

Theorem 1.2. *Let L be a field with automorphism σ of finite order r and $K = L^{\sigma}$, and assume we have representational data* ***(H)*** *for L/K as above. Given $A,B \in L[x;\sigma]$ of degrees at most d, and $\mu \in (0,1)$, there is an algorithm to compute AB with probability at least $1-\mu$, using an expected $O\tilde{}(\log(d)Sr^{\omega})$ operations in K and $O\tilde{}(\log(d)S(r+\log(1/\mu)))$ bit operations, where S is the cardinality of the set $\mathbb{S}(A,B)$ defined in* (1.1).

We also present a multivariate version of this algorithm whose cost is summarized as follows. Let $L[x_1,\ldots,x_n;\sigma_1,\ldots,\sigma_n]$ be a multivariate skew polynomial ring, with the relations $x_i a = \sigma_i(a)x_i$ and $x_i x_j = x_j x_i$, where each σ_i is an automorphism of L.

Theorem 1.3. *Let L be a field with automorphism σ of finite order r, $K = L^{\sigma}$ and $\sigma_i = \sigma^{e_i}$ for $0 \leq e_1,\ldots,e_n < r$, and assume we have the representation data* ***(H)*** *for L/K. Given $A,B \in L[x_1,\ldots,x_n;\sigma_1,\ldots,\sigma_n]$ of total degree at most D, and $\mu \in (0,1)$, there is an algorithm to compute AB with probability at least $1-\mu$, using an expected $O\tilde{}(nr^{\omega}S\log D)$ operations in K plus $O\tilde{}(n^2S\log D + nSr\log D + nS\log D\log(1/\mu) + S\log r\log(1/\mu))$ bit operations, where $S = \#\mathbb{S}(A,B)$.*

For historical perspective and comparison, algorithms to compute sparse multiplication of usual commutative polynomials has seen considerable research recently, both in theory and in practice. New algorithms for polynomials with at most t terms have been developed to keep the time proportional to the worst-case output size, $O(t^2)$, and low space complexity, both in theory and in practice [18, 22, 23]. This is particularly important for multivariate polynomials [30]. The aforementioned work of Arnold & Roche [2] adapts to the potential even smaller output size, and when the support is know [31] demonstrate greater improvements. See the excellent

[1]We thank the anonymous referee for pointing this out.

recent survey of [28] on the state of the art in sparse polynomial computation.

2 Sparse multiplication

In this section, we give a Las Vegas algorithm for the multiplication of sparse skew polynomials, proving Theorem 1.1. Our algorithm is based on Caruso and Le Borgne's [6]. As in that reference, the key operation is an evaluation-interpolation based multiplication algorithm modulo $x^r - 1$; the main difference is that the number of evaluations in our algorithm depends on the sparsity of the product. To build the main algorithm upon this special case, we will follow [6] with few modifications.

2.1 Preliminaries

2.1.1. Division modulo central elements. For a non-zero Z in the center of $L[x;\sigma]$, and for A in $L[x;\sigma]$, there are unique $Q, F \in L[x;\sigma]$ such that $A = QZ+F = ZQ+F$, with $F = 0$ or $\deg F < \deg Z$; we write $F = A \bmod Z$. This makes the canonical morphism

$$\begin{array}{cccc} \varepsilon: & L[x;\sigma] & \to & L[x;\sigma]/\langle Z\rangle, \\ & A & \mapsto & A \bmod Z, \end{array}$$

an endomorphism of K-algebras.

Since σ has order r, the equality $x^r c = \sigma^r(c)x^r = cx^r$ holds for all c in L. As a result, any polynomial of the form $Z = B(x^r)$, where $B \in K[x]$, is in the center of $L[x;\sigma]$ (actually, all central elements are of this form, but we won't need this). We will only use the very particular cases $Z = x^r - a$ and $Z = x^r - 1$, for which we have simple explicit formulas for the remainders. In particular, for the latter, if we consider a skew polynomial $C = c_1 x^{e_1} + \cdots + c_S x^{e_S} \in L[x;\sigma]$, with all c_i in L, then we have

$$C \bmod (x^r - 1) = c_1 x^{e_1 \bmod r} + \cdots + c_S x^{e_S \bmod r}, \qquad (2.1)$$

with $e_i \bmod r$ in $\{0, 1, \ldots, r-1\}$ for all i.

2.1.2. Scalar extension. Given $A, B \in L[x;\sigma]$, to compute the product AB, we first compute different reductions $AB \mod x^r - a_i$, where $a_i \in K$, then recover AB from these reductions by Chinese remainder algorithm. The number of reductions we need depends on the degree of the product AB. If it is large, as in [6], there may not be enough elements in ground field K, so we may have to replace K by an extension K'/K of sufficiently large cardinality. We write $s := [K' : K]$, and we assume that K' is given as $K[\xi]/g(\xi)$, for some degree-s irreducible $g \in K[\xi]$; in particular, all operations $+, \times, \div$ in K' take $O\tilde{~}(s)$ operations in K.

We will then define $L' := L \otimes_K K'$; L' still has dimension r over K', but it does not have to be a field; it is in general a product of fields. The extension of σ to L' is the automorphism $\sigma' := \sigma \otimes_K \mathrm{id}$; it still has order r and admits K' as its fixed set.

The K-bases $\mathscr{W} = (\omega_0, \ldots, \omega_{r-1})$ and $\mathscr{N} = (\nu_0, \ldots, \nu_{r-1})$ of L extend to K'-bases $\mathscr{W}' = (\omega'_0, \ldots, \omega'_{r-1})$ and $\mathscr{N}' = (\nu'_0, \ldots, \nu'_{r-1})$ of L', with $\omega'_i = \omega_i \otimes_K 1$ and $\nu'_i = \nu_i \otimes_K 1$ for all i. In the new working basis $\mathscr{W}'$, addition, multiplication, and the inversion of invertible elements still take $O\tilde{~}(r)$ operations $(+, \times, \div)$ in K', that is, $O\tilde{~}(rs)$ operations in K; besides, $\mathscr{N}'$ is still a normal basis. Finally, the change-of-basis matrices between $\mathscr{W}$ and $\mathscr{N}$ still describe change-of-basis between $\mathscr{W}'$ and $\mathscr{N}'$ (but now seen as matrices over K').

To summarize, changing the ground field from K to K' affects almost nothing in our setup; the only point that will require our attention is that L' may not be a field, so non-zero elements may not be invertible.

2.2 Multiplication modulo $x^r - 1$

We start with a multiplication algorithm modulo $x^r - 1$. As explained above, we suppose that we are given a field extension K'/K of degree s, and we give an algorithm for multiplication in $L'[x;\sigma']/\langle x^r - 1\rangle$.

A skew polynomial $A \in L'[x;\sigma']$ defines a K'-linear mapping $A^* : L' \to L'$ obtained by evaluating A at σ'. For a in L', we will write $A(a)$ instead of $A^*(a)$; since σ' has order r, $A(a)$ is actually well-defined for A in $L'[x;\sigma']/\langle x^r - 1\rangle$.

This suggests an evaluation / interpolation strategy for multiplication $L'[x;\sigma']/\langle x^r - 1\rangle$. This idea is already in [6], but does not take sparsity into account there; the following algorithm achieves this, by using evaluation and interpolation at a geometric progression.

We first give the overview of the algorithm, then discuss subroutines and establish their cost bounds. Below, remember that elements of L' are always represented on the working basis $\mathscr{W}'$.

Algorithm 1: Sparse multiplication modulo $x^r - 1$.

```
Input: Two polynomials $A, B \in L'[x;\sigma']/\langle x^r - 1\rangle$.
Output: The product $AB \in L'[x;\sigma']/\langle x^r - 1\rangle$.
    Step 1: Compute $\mathbb{S}_r(A,B)$ as in (1.2) and let $R = \#\mathbb{S}_r(A,B)$.
    Step 2: Compute $b_i = B(\nu_0'^{\,i})$, for $i = 0, 1, \ldots, R-1$ and let $\boldsymbol{B}$ be
        the $r \times R$ matrix over $K'$ whose $i$th column is the coefficient
        vector of $b_i$ for all $i$.
    Step 3: Compute $e_i = A(\nu'_i)$, for $i = 0, \ldots, r-1$ and let $\boldsymbol{E}$ be
        the $r \times r$ matrix over $K'$ whose $i$th column is the coefficient
        vector of $e_i$ for all $i$.
    Step 4: Compute $\boldsymbol{F} = \boldsymbol{E}\boldsymbol{M}_{\mathscr{W}\to\mathscr{N}}\boldsymbol{B}$ and let $f_0, \ldots, f_{R-1}$ be the
        elements of $L'$ whose coefficient vectors are the columns
        of $\boldsymbol{F}$.
    Step 5: Return the unique polynomial $C = \sum_{\alpha \in \mathbb{S}_r(A,B)} c_\alpha x^\alpha$
        such that $C(\nu_0'^{\,i}) = f_i$ for all $i$.
```

Proposition 2.1. *Under assumption* **H**, *Algorithm 1 computes the product AB using $O\tilde{~}(R^{\omega-2}r^2 s)$ operations in K and $O\tilde{~}(r)$ bit operations.*

Proof. Write $C = AB \in L'[x;\sigma']/\langle x^r - 1\rangle$. Since $C^* = A^* \circ B^*$, we get $C(\nu_0'^{\,i}) = A(B(\nu_0'^{\,i}))$, for $i = 0, \ldots, R-1$.

The product $\boldsymbol{E}\boldsymbol{M}_{\mathscr{W}\to\mathscr{N}}$ is by construction the matrix of $A^* : L' \to L'$ (in the working basis), and the columns of $\boldsymbol{B}$ are the coefficient vectors of $B(\nu_0'^{\,i})$, for $i = 0, \ldots, R-1$, also written in the working basis. As a result, $C(\nu_0'^{\,i}) = f_i$ holds for $i = 0, \ldots, R-1$. In view of formula (2.1), we know that the support $\mathrm{supp}(C)$ is contained in $\mathbb{S}_r(A,B)$; then, we prove in §2.2.2 that Step 5 correctly recovers C.

In terms of runtime, Step 1 takes $O\tilde{~}(r)$ bit operations (by §2.2.1 below) and Step 2 takes $O\tilde{~}(r^2 s)$ operations in K (§2.2.2). Step 3 takes $O\tilde{~}(r^2)$ operations in K' by [6, Prop. 1.6], which is also $O\tilde{~}(r^2 s)$ operations in K. The cost of Step 4 is $O\tilde{~}(R^{\omega-2}r^2 s)$ operations in K, using block matrix multiplication. Finally, Step 5 takes another $O\tilde{~}(r^2 s)$ operations in K (§2.2.2). ◻

2.2.1. Computing the sumset.

Given A and B as above, we show here how to compute the sumset $\mathbb{S}_r(A,B)$. Assume the supports of A,B are $\mathbb{S}_A,\mathbb{S}_B$, respectively, and let

$$\widetilde{A}=\sum_{d\in\mathbb{S}_A} y^d \in \mathbb{Z}[y], \quad \widetilde{B}=\sum_{d\in\mathbb{S}_B} y^d \in \mathbb{Z}[y]$$

be the commutative polynomials whose supports are $\mathbb{S}_A,\mathbb{S}_B$ and coefficients are all 1. To compute $\mathbb{S}_r(A,B)=\{(e_A+e_B) \bmod r \mid e_A\in\mathbb{S}_A, e_B\in\mathbb{S}_B\}$, it is enough to compute the support of $\widetilde{AB} \bmod (y^r-1)$. Using fast multiplication in $\mathbb{Z}[y]$, this takes $O^\sim(r)$ bit operations, as claimed.

2.2.2. Evaluation-interpolation at a geometric progression.

Let $C=c_1x^{e_1}+\cdots+c_tx^{e_t}$ be in $L'[x,\sigma']/\langle x^r-1\rangle$, with $0\le e_1<\cdots<e_t<r$. Here we show how to evaluate C at the points ${v'_0}^i$, for $i=0,1,\ldots,R-1$, for some integer R, with $t\le R\le r$; we also show how to recover C from these values, assuming $e_1,\ldots,e_t$ are known.

For $i\ge 0$, the value $C({v'_0}^i)$ is by definition $C^*({v'_0}^i)$, that is,

$$\begin{aligned} C({v'_0}^i) &= c_1\sigma^{e_1}(v'_0)^i+\cdots+c_t\sigma^{e_t}(v'_0)^i \\ &= c_1{v'_{e_1}}^i+\cdots+c_t{v'_{e_t}}^i. \end{aligned}$$

Taken all together for $i=0,\ldots,R-1$, these equalities give

$$\begin{bmatrix} C({v'_0}^0) \\ C({v'_0}^1) \\ \vdots \\ C({v'_0}^{R-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ v'_{e_1} & v'_{e_2} & \cdots & v'_{e_t} \\ \vdots & \vdots & & \vdots \\ {v'_{e_1}}^{R-1} & {v'_{e_2}}^{R-1} & \cdots & {v'_{e_t}}^{R-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix}.$$

Proposition 2.2. *Given C and R as above, with $t\le R\le r$, we can compute $C({v'_0}^i)$, for $i=0,1,\ldots,R-1$, using $O^\sim(r^2s)$ operations in K. Given $e_1,\ldots,e_t$, we can recover $c_1,\ldots,c_t$ from these values using $O^\sim(r^2s)$ operations in K as well.*

Proof. The matrix giving the values $C({v'_0}^i)$ is transposed Vandermonde, built on the conjugates v'_{e_i}. A matrix-vector product by such a matrix takes $O^\sim(\max(R,t))\in O^\sim(r)$ operations $+,\times$ in L'; this is $O^\sim(r^2)$ operations in K', and thus $O^\sim(r^2s)$ operations in K.

Conversely, to recover C, we need to solve such a system (keeping only the first t rows). This takes $O^\sim(r)$ operations $+,\times$ in L' and $O(r)$ inversions - the former add up to $O^\sim(r^2s)$ operations in K, as above. The terms we have to invert are products of the differences $v'_{e_i}-v'_{e_k}$, so they are all of the form $\alpha\otimes_K 1$, for various non-zero $\alpha\in L$, so they are all units in L'. As a result, these inversions cost a total $O^\sim(r^2)$ operations in K. □

2.3 Multiplication modulo x^r-a

This section follows closely [6, Sec. 2.1], with only a few minor differences; in particular, correctness of the procedure below is established in that reference.

Let K' and L' be as above, with $[K':K]=s$, and let λ be a unit in L'. We define the *norm*

$$a:=\lambda\sigma'(\lambda)\cdots{\sigma'}^{r-1}(\lambda),$$

Note that we know $a\in K'$ since $a=\sigma(a)$, which is why we need to extend K to K'. We now consider multiplication in $L'[x;\sigma']/\langle x^r-a\rangle$. The main idea is to reduce multiplication modulo x^r-a to multiplication modulo x^r-1. For this, define the L'-linear map $\delta: L'[x;\sigma']\to L'[x;\sigma']$ by setting $\delta(x^i)=\lambda\sigma'(\lambda)\cdots{\sigma'}^{i-1}(\lambda)x^i$. As proved in [6], it induces an L'-algebra isomorphism $\delta: L'[x;\sigma']/\langle x^r-a\rangle\to L'[x;\sigma']/\langle x^r-1\rangle$.

Algorithm 2: Multiplication modulo x^r-a.

Input:

- An element $\lambda\in L'^\times$.
- A,B in $L'[x;\sigma']/\langle x^r-a\rangle$, where $a=\lambda\sigma'(\lambda)\cdots{\sigma'}^{r-1}(\lambda)$.

Output: The product $AB\in L'[x;\sigma']/\langle x^r-a\rangle$.

```
Step 1: Compute $s_i = {\sigma'}^i(\lambda)$ for $i=0,\ldots,r-1$.
Step 2: Compute $\lambda_i = s_0\cdots s_{i-1}$ for $i=1,\ldots,r$.
Step 3: Compute $A'=\delta(A)$ and $B'=\delta(B)$.
Step 4: Compute $C'=A'B'\in L'[x;\sigma']/\langle x^r-1\rangle$ by Algorithm 1.
Step 5: Return $\delta^{-1}(C')$.
```

Before analyzing the whole procedure, we discuss the first step, computing all conjugates of λ. Reference [6] assumes that the application of σ in the working basis $\mathscr{W}$ of L takes quasi-linear time, that is, $O^\sim(r)$ operations in K; from this, we would deduce that applying σ' to an element of L' takes $O^\sim(rs)$ operations in K'. However, as noted in the introduction, we would rather not make such a strong assumption. If L is given as $L=K[z]/f(z)$, and thus $L'=K[z,\xi]/\langle f(z),g(\xi)\rangle$, given $\sigma(z \bmod f)$, von zur Gathen and Shoup's iterated Frobenius algorithm [12] allows us to compute all conjugates of λ in $O^\sim(r^2)$ operations in K', that is, $O^\sim(r^2s)$ operations in K; this is optimal, up to logarithmic factors. We now show that this is still possible, working under the assumptions of this paper.

Proposition 2.3. *Under assumption* **H**, *given λ in L', one can compute the sequence $\lambda,\sigma'(\lambda),\ldots,{\sigma'}^{r-1}(\lambda)$ using $O^\sim(r^2s)$ operations in K.*

Proof. Suppose that λ has coefficients $(\beta_0,\ldots,\beta_{r-1})$ on the working basis $\mathscr{W}'$ of L'. Under assumption **(H)**, we can compute its coefficients $(\gamma_0,\ldots,\gamma_{r-1})$ on the normal basis $\mathscr{N}'$ in $O(r^2)$ operations in K', that is, $O^\sim(r^2s)$ operations in K, by a matrix-vector product with $\boldsymbol{M}_{\mathscr{W}\to\mathscr{N}}$.

Let $\boldsymbol{L}\in K'^{r\times r}$ be the matrix whose ith column contains the coefficients of ${\sigma'}^i(\lambda)$ on the working basis $\mathscr{W}'$, and let $\boldsymbol{M}_{\mathscr{N}\to\mathscr{W}}$ be the change-of-basis matrix from $\mathscr{N}'$ to $\mathscr{W}'$. Then, we have the equality

$$\boldsymbol{L}=\boldsymbol{M}_{\mathscr{N}\to\mathscr{W}}\begin{bmatrix} \gamma_0 & \gamma_{r-1} & \cdots & \gamma_1 \\ \gamma_1 & \gamma_0 & \cdots & \gamma_2 \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{r-1} & \gamma_{r-2} & \cdots & \gamma_0 \end{bmatrix}.$$

Since the right-hand is a Hankel matrix, we can left-multiply it by a vector in $O^\sim(r)$ operations in K'. Hence the total cost to compute $\boldsymbol{L}$ is $O^\sim(r^2)$ operations in K', that is, $O^\sim(r^2s)$ operations in K. □

Corollary 2.4. *Under assumption* **H**, *Algorithm 2 computes the product AB using $O^\sim(R^{\omega-2}r^2s)$ operations in K and $O^\sim(r)$ bit operations.*

Proof. The previous proposition gives the cost of computing $s_0,\ldots,s_{r-1}$; the products $\lambda_1,\ldots,\lambda_r$ can be deduced for another $O^\sim(r^2)$ operations in K', which is $O^\sim(r^2s)$ operations in K; this gives us A' and B'. To compute their product C', Proposition 2.1 takes $O^{\sim}(R^{\omega-2}r^2s)$ operations in K and $O^{\sim}(r)$ bit operations. Finally, to recover $\delta^{-1}(C')$, we have to invert all λ_is (they are units, by assumption); this takes $O^{\sim}(r^2s)$ operations in K again. □

2.4 Main algorithm

The description of the main algorithm is essentially taken from [6], but we replace the procedure for multiplication modulo $x^r - a$ given in that reference by ours. A more minor difference is that we simplify the algorithm by not fully exploiting some properties given in [6], that would allow us to save a factor $O^{\sim}(s)$; since s will be logarithmic in the input size, this is harmless. Finally, we show how fast multipoint evaluation is actually required to obtain the claimed runtime.

To compute the product AB in $L[x;\sigma]$, we compute its image modulo central moduli of the form $x^r - a_i$, for $a_0, a_1, \ldots$ as in the previous subsection. If K is a small finite field, we may have to extend it in order to guarantee the existence of sufficiently many such moduli. Suppose that A and B have degree at most d, so that $C = AB$ has degree at most $2d$, and let $e = \lceil 2d/r \rceil + 1$; this will be the number of moduli we need.

Lemma 2.5. *Let K' be an extension of K, let Γ be a subset of K' of cardinality at least $e(e+1)r$, and let $L' = L \otimes_K K'$. Fix a basis of L' over K'. Then for $\lambda_1, \ldots, \lambda_e$ in L', with coefficients taken uniformly at random in Γ, the probability that their norms $a_1, \ldots, a_e$ be non-zero and pairwise distinct is at least $1/2$.*

Proof. For λ in L', its norm $a = \lambda\sigma'(\lambda)\cdots\sigma'^{r-1}(\lambda)$ is the determinant of the multiplication endomorphism by λ (seen as a K'-linear map $L' \to L'$). Hence, it is a non-constant homogeneous polynomial of degree r in the coefficients of λ (on an arbitrary K'-basis of L'); we write it $\Delta(\lambda)$. Then, the conclusion we want is the non-vanishing of the product of all $\Delta(\lambda_i)$ and $\Delta(\lambda_i) - \Delta(\lambda_j)$, for $1 \le i < j \le e$. This is an expression of degree $e(e+1)r/2$ in the coefficients of the λ_i's, so the conclusion follows from the DeMillo-Lipton-Schwartz-Zippel lemma. □

Algorithm 3: Multiplication.

Input: Two polynomials $A, B \in L[x;\sigma]$ of degree at most d.
Output: AB with probability at least $1/2$, or `error`

```
Step 1: Let $e = \lceil 2d/r \rceil + 1$.
Step 2: Build an extension $K'$ of $K$, such that $|K'| \ge e(e+1)r$
    and let $s = [K' : K]$.
Step 3: Pick a subset $\Gamma$ of $K'$ of cardinality at least $e(e+1)r$.
Step 4: Pick $\lambda_1, \cdots, \lambda_e$ in $L' = L \otimes_K K'$. by choosing their
    coefficients uniformly at random in $\Gamma$.
Step 5: Compute the norms $a_1, \ldots, a_e$ of $\lambda_1, \cdots, \lambda_e$. If any of
    them vanishes, raise an error.
Step 6: Compute all $A_i = A \bmod (x^r - a_i)$ and $B_i = B \bmod$
    $(x^r - a_i)$.
Step 7: Compute all $C_i = A_iB_i \bmod (x^r - a_i)$.
Step 8: Recover $C = AB$ from $C_1, \ldots, C_e$.
```

Proposition 2.6. *Under assumption* **H**, *Algorithm 3 computes the product AB using an expected $O^{\sim}(\max(d,r)rR^{\omega-2})$ operations in K and $O^{\sim}(\max(d,r))$ bit operations, with probability of success at least $1/2$; otherwise, it raises an error.*

Proof. For K finite, s is $O(\log(dr))$, and K' can be built in an expected $O(\log(dr)^2)$ operations in K [29]; if K is infinite, we take $K' = K$ and $s = 1$. Given the λ_i's, the cost of computing all a_i's will be subsumed in that of the further steps. If the conclusions of Lemma 2.5 hold, then all λ_i's are invertible (so we can apply the algorithm of the previous section), and their norms a_i's are pairwise distinct.

Write $A = \sum_{j<r} \alpha_j(x^r)x^j$, $B = \sum_{j<r} \beta_j(x^r)x^j$ and $C = AB = \sum_{j<r} \gamma_j(x^r)x^j$, where all $\alpha_j, \beta_j, \gamma_j$ have degree at most $\lceil 2d/r \rceil = e-1$. Then, for $i \le e$, $A \bmod (x^r - a_i) = \sum_{j<r} \alpha_j(a_i)x^j$. Thus, Step 6 amounts to evaluating $\alpha_0, \ldots, \alpha_{r-1}$ at $a_1, \ldots, a_e$ (and similarly for B). This takes $O^{\sim}(\max(d,r))$ operations in K' by fast evaluation, which is also $O^{\sim}(\max(d,r))$ operations in K. Step 7 involves $O(e)$ calls to Algorithm 2; this costs $O^{\sim}(R^{\omega-2}er^2s)$ operations in K and $O^{\sim}(er)$ bit operations. The former number is $O^{\sim}(\max(d,r)rR^{\omega-2})$, and the latter $O^{\sim}(\max(d,r))$. Since $\mathbb{S}_r(A,B)$ is the sumset for all reductions C_i, the computation of $\mathbb{S}_r(A,B)$ needs to be done only once, reducing the overall computing time. Finally, given $C_i = C \bmod (x^r - a_i)$, as the $x^r - a_i$ are central elements in $L'[x;\sigma]$, the reductions C_i have the same form as in the commutative ring $L'[x]$, and we can regard C, C_i all of them as in the ring $L'[x]$. For e pairwise distinct a_i, we can recover C by r interpolations in degree $e-1$ in K' for another $O^{\sim}(\max(d,r))$ operations in K. If the a_i's are not pairwise distinct, the interpolation algorithm raises an error. □

Our main algorithm now repeats the procedure above until it succeeds; this will happen after an expected $O(1)$ attempts, thereby establishing Theorem 1.1.

3 A supersparse algorithm

Let again A and B be in $L[x;\sigma]$, both of degree at most d. We now give a multiplication algorithm whose complexity is polynomial in r, $\log(d)$ and S, where S is the size of the sumset $\mathbb{S}(A,B) = \mathrm{supp}(A) + \mathrm{supp}(B)$ (recall that the support of AB is contained in $\mathbb{S}(A,B)$). The first part of the algorithm is Monte Carlo, and costs $O^{\sim}(\log(d)S\log(1/\mu))$ bit operations, for a probability of failure at most μ; the rest of the algorithm is Las Vegas.

3.0.1. Outlook of the algorithm. The first step in our algorithm computes $\mathbb{S}(A,B)$ as defined above. For any given error tolerance μ, the algorithm in [2] achieves this with bit complexity $O^{\sim}(\log(d)S\log(1/\mu))$ and with probability at least $1-\mu$. Here, and in what follows, we write $S = \#\mathbb{S}(A,B)$.

Let us write $\mathbb{S}(A,B) = \{e_1, \ldots, e_S\}$ and $AB = c_1x^{e_1} + \cdots + c_Sx^{e_S} \in L[x;\sigma]$, with all c_i in L. For a non-zero multiple q of r, $x^q - 1$ is central, and we have

$$(AB) \bmod (x^q - 1) = c_1x^{e_1 \bmod q} + \cdots + c_Sx^{e_S \bmod q},$$

with $e_i \bmod q$ in $\{0, 1, \ldots, q-1\}$ for all i. If all $e_i \bmod q$ are pairwise distinct, and if we assume that $\mathbb{S}(A,B)$ is known, computing $(AB) \bmod (x^q - 1)$ allows us to recover AB.

However, even through randomization, we are not able to find a q satisfying such a condition and of growth rate less than quadratic in S. Instead, we use an approach coming from [1]: we allow for a certain number of $e_i \bmod q$ to coincide. We will then take q of the form $q = pr$, with p a prime whose size is well controlled. For p satisfying certain luckiness conditions, we will be able to recover at least half the terms in AB; then, we compute the remaining terms recursively.

3.0.2. Finding a prime. Let n be a non-zero integer, and let $\mathbb{T}$ be a subset of $\{0,\dots,2d\}$. An element e in $\mathbb{T}$ is called a *collision* modulo n if there exists $e' \neq e$ in $\mathbb{T}$ such that $e \equiv e' \bmod n$.

Lemma 3.1. *One can find using an expected $O\tilde{~}(\log(d)T)$ bit operations a prime p such that $p \in O(\log(d)T)$ and $\mathbb{T}$ has at most $T/2$ collisions modulo p, with $T = \#\mathbb{T}$.*

Proof. Let $\lambda = \max(21, \lceil 20(T-1)\ln(2d)/3\rceil)$. Then, Lemma 8 in [1] shows that if p is a random prime in $\{\lambda,\dots,2\lambda\}$, with probability at least $1/2$, $\mathbb{T}$ has less than $T/2$ collisions modulo p. In particular, trying an expected $O(1)$ such primes is sufficient to find a suitable one. By sieving, we can compute all primes up to 2λ in $O\tilde{~}(\log(d)T)$ bit operations. Given a prime p in $\{\lambda,\dots,2\lambda\}$, we can compute $\mathbb{T} \bmod p$ in the same asymptotic cost. Counting collisions can be done by (for instance) sorting all $e_i \bmod p$, in $O\tilde{~}(T\log\log(d))$ bit operations. $\square$

3.0.3. Finding half the terms. Our main procedure is the following. In addition to A and B, we take as input an "approximation" P of the product AB; as output, we return a better approximation of AB, as specified below.

Algorithm 4: Half multiplication.
Input:

- $A, B \in L[x;\sigma]$ of degrees at most d
- $P \in L[x;\sigma]$, such that all terms of P are terms of AB
- a set $\mathbb{T} \subset \{0,\dots,2d\}$ containing the support of $AB - P$

Output:

- P^* in $L[x;\sigma]$ such that all terms of P^* are terms of AB.
- a set $\mathbb{T}^* \subset \{0,\dots,2d\}$ containing the support of $AB - P^*$, such that $\#\mathbb{T}^* \leq \#\mathbb{T}/2$.

```
Step 1: find a prime $p \in O(\log(d)T)$ such that $\mathbb{T}$ has at most
    $T/2$ collisions modulo $p$, with $T = \#\mathbb{T}$.
Step 2: compute $u = (AB - P) \bmod (x^{pr} - 1)$.
Step 3: compute $f_1 = e_1 \bmod pr, \dots, f_T = e_T \bmod pr$.
Step 4: let $\mathbb{T}^* \subset \mathbb{T}$ be the set of collisions in $\mathbb{T}$ modulo $pr$.
Step 5: Let $P^* = P$. For $i = 1,\dots,T$, if $e_i$ is not in $\mathbb{T}^*$, find the
    coefficient $c_i$ of $x^{f_i}$ in $u$ and let $P^* = P^* + c_i x^{e_i}$.
Step 6: Return $P^*$ and $\mathbb{T}^*$.
```

Proposition 3.2. *Algorithm 4 is correct. Under assumption* **H**, *it uses an expected $O\tilde{~}(\log(d)Tr^{\omega})$ operations in K and $O\tilde{~}(\log(d)Tr)$ bit operations, with $T = \#\mathbb{T}$.*

Proof. By construction, all terms in P^* are either terms in P (in which case they are terms in AB), or terms in $AB - P$ (and thus in AB as well); they are thus always terms in AB, which shows that the first item holds.

Next, take a term in AB but not in P^*; then, it belongs to $\mathbb{T}$, but not to $\mathbb{T} - \mathbb{T}^*$; this proves that the support of $AB - P^*$ is in $\mathbb{T}^*$, as claimed. Finally, since $\mathbb{T}$ has at most $T/2$ collisions modulo p, it has at most $T/2$ collisions modulo pr; hence, we have $\#\mathbb{T}^* \leq \#\mathbb{T}/2$. Correctness is proved.

Next, we analyze the cost of this procedure. By Lemma 3.1, Step 1 takes an expected $O\tilde{~}(\log(d)T)$ bit operations. At Step 2, we compute u by reducing A and B modulo $x^{pr} - 1$, multiplying the remainders and reducing the product, and subtracting $P \bmod (x^{pr} - 1)$.

Since p is $O\tilde{~}(\log(d)T)$, using Theorem 1.1, the cost of computing the product modulo $x^{pr} - 1$ is an expected $O\tilde{~}(\log(d)Tr^{\omega})$ operations in K and $O(\log(d)Tr)$ bit operations. This dominates the cost of the other steps. $\square$

3.0.4. Main algorithm. The main procedure calls Algorithm 4 on rapidly decreasing supports $\mathbb{T}$; it finishes after $O(\log S)$ iterations, where S is the cardinality of $\mathbb{S}(A,B) = \mathrm{supp}(A) + \mathrm{supp}(B)$.

Algorithm 5: Multiplication.
Input:

- A, B in $L[x;\sigma]$ of degrees at most d
- error tolerance μ.

Output: with probability at least $1 - \mu$, the product AB.

```
Step 1: compute $\mathbb{S}(A,B) = \mathrm{supp}(A) + \mathrm{supp}(B)$.
Step 2: let $P = 0$ and $\mathbb{T} = \mathbb{S}(A,B)$.
Step 3: while $\mathbb{T}$ is not empty do
    a: let $P, \mathbb{T} =$ Half multiplication$(A,B,P,\mathbb{T})$.
Step 4: return $P$.
```

The following proposition results directly from Proposition 3.2, using the algorithm of Arnold and Roche [2] for computing $\mathbb{S}(A,B)$ with a cost of $O\tilde{~}(S\log(d)\log\frac{1}{\mu})$ bit operations. It establishes Theorem 1.2.

Proposition 3.3. *Algorithm 5 succeeds with probability at least $1-\mu$. Under assumption* **H**, *it uses an expected $O\tilde{~}(\log(d)Sr^{\omega})$ operations in K and an expected $O\tilde{~}(\log(d)S(r + \log(1/\mu)))$ bit operations, with $S = \#\mathbb{S}(A,B)$.*

4 Multivariate skew polynomials

Finally, we extend our second univariate multiplication algorithm to certain multivariate cases, using Kronecker substitution. One may also use the algorithm of Section 2, but the result would be exponential in the number n of variables: the runtime of the algorithm of Section 2 is polynomial in the input degree, and Kronecker substitution produces univariate polynomials of degree exponential in n.

Multivariate skew polynomials have not been as intensively studied; refer to [13, 20, 21] for recent work. Let $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]$ be a multivariate skew polynomial ring, where L is a field, with the relations $x_i a = \sigma_i(a)x_i$ and $x_i x_j = x_j x_i$ for all i,j, and where each σ_i is an automorphism of L.

In [21], using a matrix of endomorphisms, the authors define more general multivariate skew polynomials. Our definition seems to correspond to a diagonal matrix containing automorphisms, which is only a special case of the definition in [21]. However, in [21], x_i, x_j do not commute for $i \neq j$, which is used to make sure the uniqueness of evaluation, defined as the remainder of a right division. In our definition, we assume $x_i x_j = x_j x_i$ for all i,j, and the evaluation at a point is defined as the value of function which replaces x_i in the skew polynomial with σ_i.

We assume that there exists an automorphism σ of L, having order r, and integers $e_1,\dots,e_n$ such that for all i, $\sigma_i = \sigma^{e_i}$, and as before we let K be the fixed field of σ. This assumption is for instance valid when L is a finite field.

Consider integers $\boldsymbol{N} = (N_1, \dots, N_n)$ and the L-linear mapping $\Psi_{\boldsymbol{N}}$ defined by

$$\begin{array}{rccc} \Psi_{\boldsymbol{N}}: & L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n] & \to & L[x;\sigma] \\ & x_1^{d_1}\cdots x_n^{d_n} & \mapsto & x^{d_1N_1+\cdots+d_nN_n} \end{array}$$

This is simply a Kronecker substitution, in a non-commutative setting.

Lemma 4.1. *If $N_i \equiv e_i \bmod r$ for all i, then $\Psi_{\boldsymbol{N}}$ is a K-algebra morphism.*

Proof. Since $\Psi_{\boldsymbol{N}}$ acts multiplicatively on monomials, the only property we have to verify is that for integers $(d_1, \dots, d_n)$ and b in L, $\Psi_{\boldsymbol{N}}(x_1^{d_1}\cdots x_n^{d_n})\Psi_{\boldsymbol{N}}(b) = \Psi_{\boldsymbol{N}}(x_1^{d_1}\cdots x_n^{d_n}b)$. The former equals $\sigma^{\sum_{i=1}^n d_iN_i}(b)x^{\sum_{i=1}^n d_iN_i}$, while the latter is $\sigma^{\sum_{i=1}^n d_ie_i}(b)x^{\sum_{i=1}^n d_iN_i}$. Our assumption implies that the exponents $\sum_{i=1}^n d_iN_i$ and $\sum_{i=1}^n d_ie_i$ are the same modulo r, and the conclusion follows. □

For $D \geq 0$, let $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]_D$ be the L-vector space of skew polynomials of total degree less than D. We now discuss conditions on $\boldsymbol{N}$ that ensures that the restriction of $\Psi_{\boldsymbol{N}}$ to $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]_D$ is injective.

Lemma 4.2. *Let D be a positive integer. Assume $N_i \in \mathbb{N}_{>0}$ satisfy $D \leq N_1$ and $N_iD \leq N_{i+1}$ for $1 \leq i < n$. Then the restriction of $\Psi_{\boldsymbol{N}}$ to $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]_D$ is injective.*

Proof. Suppose $m \in \mathbb{N}_{>0}$ can be represented as $m = \sum_{i=1}^n d_iN_i$ with $\sum_{i=1}^n d_i < D$, and in particular $0 \leq d_i < D$. It suffices to show that this relation defines d_n uniquely; once this is known, we set $m' = m - d_nN_n$ and the claim follows by induction. Precisely, we prove that $d_n = \lfloor \frac{m}{N_n} \rfloor$. Since

$$\begin{aligned} d_nN_n \leq m = \sum_{i=1}^n d_iN_i &\leq (D-1)(1+\sum_{i=1}^{n-1} N_i) + d_nN_n \\ &= D(1+\sum_{i=1}^{n-1}N_i) - (1+\sum_{i=1}^{n-1}N_i) + d_nN_n \\ &\leq (D+\sum_{i=2}^{n}N_i) - (1+\sum_{i=1}^{n-1}N_i) + d_nN_n \\ &= D - 1 - N_1 + N_n + d_nN_n \\ &< N_n + d_nN_n = (d_n+1)N_n. \end{aligned}$$

Dividing by N_n on both sides, we get $d_n \leq \frac{m}{N_n} < (d_n+1)$. Since d_n is an integer, we get $d_n = \lfloor \frac{m}{N_n} \rfloor$, and we are done. □

The following algorithm describes how to compute the d_i's.

Algorithm 6: Index.

Input:

- Positive integers $N_1,\dots,N_n$, where $D \leq N_1$ and $N_iD \leq N_{i+1}$ for $i = 1,\dots,n-1$.
- A positive integer $m = d_1N_1 + \cdots + d_nN_n$, where $0 \leq d_i < D$ for $i = 1,\dots,n$.

Output: The indices $d_1,\dots,d_n$.

```
Step 1: For $i = n,\dots,1$ do
    a: Let $d_i = \lfloor \frac{m}{N_i} \rfloor$.
    b: Let $m = m - d_iN_i$.
Step 2: Return $d_1,\dots,d_n$.
```

Lemma 4.3. *Algorithm 6 is correct and requires $O^{\sim}(n\log(D) + n\log(N_n))$ bit operations.*

Proof. Correctness comes from the expression $d_n = \lfloor \frac{m}{N_n} \rfloor$, which was established in the proof of Lemma 4.2. As to complexity, each iteration of Step 1 costs a constant number of arithmetic operations. Since $m \leq DN_n$ and $N_1 < N_2 < \cdots < N_n$, the height of m is $O(\log(D) + \log(N_n))$, and the total cost of Step 1 is $O^{\sim}(n\log(D) + n\log(N_n))$ bit operations. □

Taking into account the constraints in the two previous lemmas, we obtain the following construction of integers $N_1,\dots,N_n$.

Lemma 4.4. *Given a positive integer D, set $N_0 = 1$ and define $\boldsymbol{N} = (N_1,\dots,N_n)$ recursively by*

$$N_{i+1} = e_{i+1} + k_{i+1}r, \text{ where } k_{i+1} = \max\{\lceil \frac{DN_i - e_{i+1}}{r} \rceil, 0\}.$$

Then $\boldsymbol{N}$ satisfies the conditions of Lemmas 4.1 and 4.2, and $N_i \leq rD^{n+1}$ holds for all i.

Proof. The congruence conditions clearly hold. For $i \geq 0$, we claim that $N_iD \leq N_{i+1} \leq N_iD + r$; the left-hand side then proves the inequalities needed in Lemma 4.2.

If $k_{i+1} = 0$, then $N_iD \leq e_{i+1}$, so $N_{i+1} = e_{i+1}$, which means $N_iD \leq N_{i+1}$. On the other hand, since $0 \leq e_{i+1} < r$, we have $N_{i+1} \leq N_iD + r$. If $k_{i+1} > 0$, then $k_{i+1} = \lceil \frac{DN_i - e_{i+1}}{r} \rceil$, so we have $\frac{DN_i - e_{i+1}}{r} \leq k_{i+1} < \frac{DN_i - e_{i+1}}{r} + 1$. This gives $DN_i - e_{i+1} \leq k_{i+1}r < DN_i - e_{i+1} + r$, and thus $DN_i \leq N_{i+1} < DN_i + r$. In either case, we proved the claim. This inequalities also imply (by induction) that all N_i's satisfy $N_i \leq D^i + r(D^i - 1)/(D-1)$, and thus $N_i \leq rD^{n+1}$. □

Corollary 4.5. *Let D and $\boldsymbol{N}$ as in the previous lemma, and let C be in $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]_D$, with $\#C \leq S$. Given $\Psi_{\boldsymbol{N}}(C)$ we can recover C in $O^{\sim}(Sn^2\log(D) + Sn\log(r))$ bit operations.*

Proof. Apply Algorithm 6 to all terms of $\Psi_{\boldsymbol{N}}(C)$. Each instance takes $O^{\sim}(n\log(D) + n\log(N_n))$ bit operations, and the previous lemma proved that $\log(N_n)$ is $O(n\log(D) + \log(r))$. □

We can now present our sparse multivariate multiplication algorithm.

Algorithm 7: Multivariate Multiplication.

Input:

- A, B in $L[x_1,\dots,x_n;\sigma_1,\dots,\sigma_n]$
- error tolerance μ

Output: with probability at least $1-\mu$, the product AB

```
Step 1: let $D = \deg A + \deg B + 1$.
Step 2: let $\boldsymbol{N}$ be as in Lemma 4.4.
Step 3: compute $\widetilde{A} = \Psi_{\boldsymbol{N}}(A)$ and $\widetilde{B} = \Psi_{\boldsymbol{N}}(B)$
Step 4: compute $\widetilde{C} = \widetilde{A}\widetilde{B}$ by calling Algorithm 5 with inputs $\widetilde{A}$, $\widetilde{B}$ and $\mu$
Step 5: return $\Psi_{\boldsymbol{N}}^{-1}(\widetilde{C})$
```

Proposition 4.6. *Algorithm 7 computes AB with probability at least $1-\mu$ and costs $O^{\sim}(nr^{\omega}S\log D)$ field operations in K plus $O^{\sim}(n^2S\log D + nSr\log D + nS\log D\log(1/\mu) + S\log r\log(1/\mu))$ bit operations, where $S = \#\mathbb{S}(A,B)$.*

Proof. Correctness comes from Lemma 4.4: if the product $\widetilde{AB}$ computed in Step 4 is correct, then the output is the product AB. By Proposition 3.3, Algorithm 5 returns the correct product with probability at least $1-\mu$, so we are done.

Step 2 needs n operations. Since the bit-lengths are $O(n\log D + \log r)$, the bit cost is $O^{\sim}(n^2\log D + n\log r)$. At Step 3, since $\#A, \#B \leq S$, we use at most $O^{\sim}(n^2S\log D + nS\log r)$ bit operations. At Step 4, the degree of $\widetilde{f}\cdot\widetilde{g}$ is $\widetilde{d}$, so by Proposition 3.3 we use $O^{\sim}(r^{\omega}S\log(\widetilde{d}))$ operations in K and $O^{\sim}(\log(\widetilde{d})S(r+\log(1/\mu)))$ bit operations. Since $\widetilde{d} \leq DN_n$ and N_n is in $O(rD^n)$, this is $O^{\sim}(nr^{\omega}S\log D)$ operations in K and $O^{\sim}(nSr\log D + S\log r\log(1/\mu) + nS\log D\log(1/\mu))$ bit operations. In Step 5, by Lemma 4.3, we use $O^{\sim}(nS\log D + nS\log N_n) = O^{\sim}(n^2S\log D + nS\log r)$ bit operations. □

5 Conclusions

In this paper, we present new multiplication algorithms for skew polynomials. Our first new algorithm is a Las Vegas algorithm for multiplication in $L[x;\sigma]$; the second algorithm is for multiplication of "supersparse" polynomials in $L[x;\sigma]$. Its cost is sensitive to the number of non-zero terms, and is significantly faster than previous algorithms when the product has large degree but few terms.

Finally, we consider multiplying sparse multivariate skew polynomials in $L[x_1,\ldots,x_n;\sigma_1,\ldots,\sigma_n]$. We introduced a non-commutative Kronecker substitution scheme, and present an algorithm with polynomial runtime in the input and output size. This is a particular improvement over standard dense algorithms, which could be of exponential complexity in the number of non-zero input terms.

Acknowledgement

The authors would like to acknowledge the careful anonymous review of this paper.

References

[1] A. Arnold, M. Giesbrecht, and D. Roche. 2013. Faster sparse interpolation of straight-line programs. In *International Workshop on Computer Algebra in Scientific Computing*. Springer, 61–74.
[2] A. Arnold and D. Roche. 2015. Output-sensitive algorithms for sumset and sparse polynomial multiplication. In *ISSAC'15*. ACM Press, 29–36.
[3] D. Boucher, P. Gaborit, W. Geiselmann, O. Ruatta, and F. Ulmer. 2010. Key exchange and encryption schemes based on non-commutative skew polynomials. In *International Workshop on Post-Quantum Cryptography*. Springer, 126–141.
[4] D. Boucher, W. Geiselmann, and F. Ulmer. 2007. Skew-cyclic codes. *Applicable Algebra in Engineering, Communication and Computing* 18, 4 (2007), 379–389.
[5] D. Boucher and F. Ulmer. 2009. Coding with skew polynomial rings. *Journal of Symbolic Computation* 44, 12 (2009), 1644–1656.
[6] X. Caruso and J. Le Borgne. 2017. Fast multiplication for skew polynomials. In *ISSAC'17*. ACM, 77–84.
[7] D. Coppersmith and S. Winograd. 1990. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* 9, 3 (1990), 251–280.
[8] J.-M. Couveignes and R. Lercier. 2009. Elliptic periods for finite fields. *Finite Fields Their Appl.* 15, 1 (2009), 1–22.
[9] E. Gabidulin. 1985. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii* 21, 1 (1985), 3–16.
[10] F. Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *ISSAC'14*. ACM Press, 296–303.
[11] J. von zur Gathen and M. Giesbrecht. 1990. Constructing normal bases in finite fields. *J. Symb. Comput* 10 (1990), 547–570.
[12] J. von zur Gathen and V. Shoup. 1992. Computing Frobenius maps and factoring polynomials. *Computational Complexity* 2, 3 (1992), 187–224.
[13] W. Geiselmann and F. Ulmer. 2019. Skew Reed-Muller codes. *Contemporary mathematics* (2019), 107–116.
[14] M. Giesbrecht. 1998. Factoring in skew-polynomial rings over finite fields. *Journal of Symbolic Computation* 26, 4 (1998), 463–486.
[15] M. Giesbrecht, A. Jamshidpey, and É Schost. 2019. Quadratic-Time Algorithms for Normal Elements. In *ISSAC'19*. ACM Press, 179–186.
[16] K. Girstmair. 1999. An algorithm for the construction of a normal basis. *Journal of Number Theory* 78, 1 (1999), 36–45.
[17] D. Goss. 1996. *Basic Structures of Function Field Arithmetic*. Springer Berlin Heidelberg.
[18] S. Johnson. 1974. Sparse polynomial arithmetic. *ACM SIGSAM Bulletin* 8, 3 (1974), 63–71.
[19] E. Kaltofen and V. Shoup. 1998. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp.* 67, 223 (1998), 1179–1197.
[20] U. Martínez-Penas. 2019. Classification of multivariate skew polynomial rings over finite fields via affine transformations of variables. *arXiv: 1908.06833* (2019).
[21] U. Martínez-Penas and F. R. Kschischang. 2019. Evaluation and interpolation over multivariate skew polynomial rings. *Journal of Algebra* 525 (2019), 111–139.
[22] M. Monagan and R. Pearce. 2009. Parallel Sparse Polynomial Multiplication Using Heaps. In *ISSAC'09*. 263–269.
[23] M. Monagan and R. Pearce. 2011. Sparse Polynomial Pseudo Division Using a Heap. *J. Symb. Comp.* 46, 7 (2011), 807–822.
[24] O. Ore. 1933. Theory of non-commutative polynomials. *Annals of Mathematics* (1933), 480–508.
[25] S. Puchinger and A. Wachter-Zeh. 2016. Sub-quadratic decoding of Gabidulin codes. In *2016 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2554–2558.
[26] S. Puchinger and A. Wachter-Zeh. 2018. Fast operations on linearized polynomials and their applications in coding theory. *Journal of Symbolic Computation* 89 (2018), 194–215.
[27] F. Kschischang R. Koetter. 2008. Coding for errors and erasures in random network coding. *IEEE Transactions on Information Theory* 54, 8 (2008), 3579–3591.
[28] D. Roche. 2018. What can (and can't) we do with sparse polynomials?. In *ISSAC'18*. 25–30.
[29] V. Shoup. 1994. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation* 17, 5 (1994), 371–391.
[30] J. van der Hoeven and G. Lecerf. 2012. On the Complexity of Multivariate Blockwise Polynomial Multiplication. In *ISSAC'12*. 211–218.
[31] J. van der Hoeven and G. Lecerf. 2013. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Computation* 50 (2013), 227–254.
[32] Y. Zhang. 2010. A secret sharing scheme via skew polynomials. In *2010 International Conference on Computational Science and Its Applications*. IEEE, 33–38.