# Sparse Multiplication of Multivariate Linear Differential Operators

Mark Giesbrecht
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, On, Canada
mwg@uwaterloo.ca

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

Éric Schost
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, On, Canada
eschost@uwaterloo.ca

## ABSTRACT

We propose a new randomized algorithm for multiplication in the ring of non-commutative polynomials $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$, where $\delta_i = x_i \frac{\partial}{\partial x_i}$, dedicated to sparse inputs. The complexity of our algorithm is polynomial in the input size and in an *a priori* sparsity bound for the output.

## CCS CONCEPTS

• **Computing methodologies → Symbolic and algebraic algorithms**.

## KEYWORDS

Sparse multiplication, Linear differential operators, Sparse polynomial interpolation, Derivatives, Monte Carlo algorithm

**ACM Reference Format:**
Mark Giesbrecht, Qiao-Long Huang, and Éric Schost. 2021. Sparse Multiplication of Multivariate Linear Differential Operators. In *2021 Int'l Symposium on Symbolic & Algebraic Computation (ISSAC'2021), July 18–22, 2021, Russia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.114

## 1 INTRODUCTION

Multiplication in a (commutative) polynomial algebra $\mathbb{A}[x]$, for a ring $\mathbb{A}$, has been intensively studied; the best general result to date is Cantor and Kaltofen's FFT multiplication algorithm, which computes the product of two polynomials of degree $d$ in $\mathbb{A}[x]$ using $O(d \log d \log\log d)$ operations in $\mathbb{A}$ [6].

The question of multiplying *sparse* commutative polynomials, and the related problem of sparse interpolation, have also seen considerable interest recently, both in theory and in practice; see for instance [1, 13–15], as well as the recent survey [21] on the state of the art in sparse polynomial computations. Sparse multivariate polynomial interpolation will play an important role in the framework of this paper.

We recently presented an algorithm for the multiplication of sparse *skew polynomials*, that is, polynomials with coefficients in a field $\mathbb{K}$, subject to the commutation rule $xc = \sigma(c)x$ for $c$ in $\mathbb{K}$, for a certain automorphism $\sigma$ of $\mathbb{K}$ [11]. The techniques were in part inspired by an algorithm by Caruso and Le Borgne for the multiplication of dense skew polynomials [7].

In this paper, we turn our attention to the multiplication of sparse *differential operators*; precisely, we work in the non-commutative ring $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$, for some field $\mathbb{K}$, where for all $i$ the symbols $x_i$ and $\delta_i$ are subject to the commutation relation $\delta_i x_i = x_i\delta_i + x_i$ (all other pairs commute). In other words, $\delta_i$ is meant to represent the Euler operator $x_i\partial/\partial x_i$.

Our algorithm is based on an evaluation-interpolation scheme which was first introduced by van der Hoeven [12] in the dense univariate case. The key idea of van der Hoeven's algorithm is to evaluate linear differential operators at powers of $x$. He proved that the evaluations needed for the algorithm can be obtained by computing the product of two matrices; as a result, in the univariate case, multiplication of two linear differential operators of degree $d$ in $x$ and degree $d$ in $\delta$ can be reduced to $d \times d$ matrix multiplication. In 2008, Bostan, Chyzak and Le Roux [5] proved that the converse statement is also true in characteristic zero: they showed that matrix multiplication in size $d \times d$ reduces to the product of two operators of bidegree at most $(d, d)$ in $(x, \delta)$.

In this paper, we rely on "soft-Oh" notation: $O^{\sim}(\phi) := O(\phi \cdot \mathrm{polylog}(\phi))$, for any function $\phi$, where polylog means $\log^c$ for some fixed $c > 0$. Going beyond the "square" case, in 2012, Benoit, Bostan and van der Hoeven [4] gave a quasi-optimal multiplication algorithm for linear differential operators. They showed that over a field $\mathbb{K}$ of characteristic zero, operators in $\mathbb{K}[x, \partial]$ of bidegree less than $(d, r)$ in $(x, \partial)$ can be multiplied using $O^{\sim}(dr \min(d, r)^{\omega-2})$ operations in $\mathbb{K}$. Here $x$ and $\partial$ satisfy the commutation rule $\partial x = x\partial + 1$ (so that $\partial$ stands for differentiation with respect to $x$) and $\omega$ is such that square matrix multiplication in $\mathbb{K}^{s\times s}$ can be done in $O(s^\omega)$ operations in $\mathbb{K}$; the best known value to date is $\omega \le 2.373$ [8, 10].

In many cases, in practice, linear operators or their products are sparse, in the sense that the number of non-zero terms is small compared to the maximal possible support in a given bidegree. Hence, in this paper, our question is the following. We consider two linear differential operators $A, B$ in $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$, and assume $A, B$ are stored in the sparse representation, *i.e.* only storing the non-zero terms. Given an upper bound on the sparsity of the product $P = AB$, how to compute this product, in the sparse representation?

Many results in this paper can be established over an arbitrary field $\mathbb{K}$, with a few restrictions on its size and characteristic ($\mathbb{K}$


*ISSAC' 21, July 18–22, 2021, Saint Petersburg, Russia*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.114

should have sufficiently many elements and characteristic zero, or large enough; this is explained in detail in the next sections).

The key of this paper is to use differentiation to "embed" the exponents into the field of coefficients. The idea of differentiation comes from [2, 16]; that of embedding the exponents in $\mathbb{K}$ is from [1, 14]. We will write our operators as

$$P = \sum_{i=1}^{T_P} p_i \boldsymbol{X}^{\boldsymbol{d}_i} \boldsymbol{\Delta}^{\boldsymbol{r}_i} \tag{1}$$

with all coefficients $p_i$ in $\mathbb{K}^*$, using the symbols $\boldsymbol{d}_i = (d_{i,1}, \ldots, d_{i,n})$, $\boldsymbol{r}_i = (r_{i,1}, \ldots, r_{i,n})$ and

$$\boldsymbol{X}^{\boldsymbol{d}_i} \boldsymbol{\Delta}^{\boldsymbol{r}_i} = x_1^{d_{i,1}} \cdots x_n^{d_{i,n}} \delta_1^{r_{i,1}} \cdots \delta_n^{r_{i,n}}.$$

If, in the expression above, we have $d_{i,j} < d$ and $r_{i,j} < r$ throughout, for some integers $d$ and $r$, we say $P$ has *bidegree* less than $(d, r)$.

The quantity $T_P$ directly controls the memory requirements for storing $P$ in sparse representation, which involves storing a list of the $T_P$ pairs of non-zero coefficients $p_i$ and corresponding exponents $(\boldsymbol{d}_i, \boldsymbol{r}_i)$. Since each exponent $(\boldsymbol{d}_i, \boldsymbol{r}_i)$ occupies $O(n \log(rd))$ bits, storing $P$ in this manner involves a list of $T_P$ elements in $\mathbb{K}$ and $O(T_P\, n \log(rd))$ bits.

Collecting all terms with same exponent in $\boldsymbol{X}$, we can also define polynomials $P_{\boldsymbol{d}}$ as follows: for $\boldsymbol{d}$ in $[0, d)^n$, write

$$P_{\boldsymbol{d}} = \sum_{i \in \{1,\ldots,T_P\} \text{ s.t. } \boldsymbol{d}_i = \boldsymbol{d}} p_i \boldsymbol{Z}^{\boldsymbol{r}_i} \in \mathbb{K}[z_1, \ldots, z_n], \tag{2}$$

where $z_1, \ldots, z_n$ are commutative variables (that will be used as placeholders for $\delta_1, \ldots, \delta_n$), and where we write $\boldsymbol{Z}^{\boldsymbol{r}} = z_1^{r_1} \cdots z_n^{r_n}$ as above. Only finitely many such polynomials are non-zero; we write them $P_{\boldsymbol{d}_1}, \ldots, P_{\boldsymbol{d}_\rho}$, for some integer $\rho$ (all $\boldsymbol{d}_i$ being taken pairwise distinct). It follows that we can write

$$P = \sum_{k=1}^{\rho} \boldsymbol{X}^{\boldsymbol{d}_k} P_{\boldsymbol{d}_k}(\delta_1, \ldots, \delta_n). \tag{3}$$

This allows us to define the *$\delta$-sparsity* of $P$ as the number

$$\tau = \max\{\#P_{\boldsymbol{d}_k} \mid 1 \le k \le \rho\},$$

where $\#P_{\boldsymbol{d}_k}$ denotes the number of non-zeros terms in $P_{\boldsymbol{d}_k}$. This quantity will play an important role in the cost analyses of our algorithms.

To give runtime estimates for algorithms over an abstract field $\mathbb{K}$, we will use a mixed algebraic / boolean complexity model, counting both arithmetic operations in $\mathbb{K}$ and bit operations, the latter originating mainly from exponent manipulations. However, the algorithms rely on univariate polynomial root-finding over $\mathbb{K}$, for which no general complexity result is known, and may also require extending $\mathbb{K}$. Hence, for simplicity, in this introduction, we present our results for finite $\mathbb{K}$ only, that is, for $\mathbb{K} = \mathbb{F}_q$. In this case, since all arithmetic operations in $\mathbb{F}_q$ can be done in $O^{\sim}(\log(q))$ bit operations, it is possible to forgo breaking down the cost in algebraic / boolean parts, and only give the total cost in bit operations.

In the following theorem, $T_A$ and $T_B$ are the number of terms in the sparse representation of the inputs $A$ and $B$. The algorithm is randomized; we assume that we can obtain a random bit with bit-cost $O(1)$.

Theorem 1.1. *Let $P, A, B$ be in $\mathbb{F}_q[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$, with $P = AB$, and suppose that the following holds:*

- *$P, A, B$, have bidegree less than $(d, r)$;*
- *$r \le \mathrm{char}(\mathbb{F}_q)$.*

*There is an algorithm that takes as input $A$, $B$ and an upper bound $\tau$ on the $\delta$-sparsity of $P$, and returns $P = AB$ with probability at least $\frac{3}{4}$ using an expected*

$$O^{\sim}(n\tau T_A T_B \log^2(rd) \log^2(q))$$

*bit operations.*

Remark 1.2. *The input size is $O(n(T_A + T_B)\log(rd) + (T_A + T_B)\log(q))$ bits, so that the cost of our algorithm is polynomial in the input size and linear in the bound $\tau$.*

In general, it is of course not obvious to give a sharp bound $\tau$ on the $\delta$-sparsity of $P$ in advance. One situation where this might be possible is when working with $A, B$ in $\mathbb{Z}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$. In this case, we can compute $AB$ using Chinese Remaindering techniques, multiplying $A$ and $B$ modulo sufficiently many primes $p_1, p_2, \ldots$ For all but finitely many primes $p$, $AB$ mod $p$ has the same number of terms as $AB$, so after computing $AB$ mod $p_1$ (by any other algorithm), we can use the $\delta$-sparsity of this product as a bound for all others $AB$ mod $p_i$ (of course, failure is possible, and should be quantified).

We may also estimate the upper bound of $\tau$ using early termination as in [18], to probabilistically detect when we have enough queries to interpolate the coefficients $P_{\boldsymbol{d}_k}$. This technique is simple and efficient. As an alternative direction, it might be possible to extend the approach used in Arnold and Roche's "output sensitive" algorithm for the multiplication of sparse commutative polynomials [1]. Finally in the univariate case, that is, with $n = 1$, we always have the upper bound $\tau \le r + 1$. In this case, the runtime of our algorithm is an expected $O^{\sim}(rT_A T_B \log^2(d) \log^2(q))$ bit operations.

Table 1 gives a comparison of our results with other multiplication algorithms in the univariate case. Note that the costs of other algorithms are counting arithmetic operations in fields of characteristic zero, whereas ours is measured in boolean complexity over a finite field.

**Table 1: A "soft-Oh" comparison of multiplication**

| Algorithm | Complexity |
|---|---|
| Iterative schemes [5] | $dr \min(d, r)$ |
| Takayama [5] | $dr \min(d, r)$ |
| van der Hoeven ($d = r$) [12] | $d^\omega$ |
| Benoit-Bostan-van der Hoeven [4] | $dr \min(d, r)^{\omega-2}$ |
| Naive sparse | $rT_A T_B$ |
| This paper ($\tau$ is known) | $\tau T_A T_B \log^2(rd) \log^2(q)$ |

*Organization of the paper.* In Section 2, we first present a sparse interpolation algorithm over the commutative polynomial ring $\mathbb{K}[z_1, \ldots, z_n]$. It is inspired by Prony's algorithm, but bypasses discrete logarithms, at the cost of a slightly more extensive input. In Section 3, we use this result to design an interpolation algorithm for sparse linear differential operators in $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$. In

Section 4, combining the interpolation algorithm of Section 3 and fast evaluation techniques, we propose our sparse multiplication algorithm for linear differential operators in $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$; we also briefly discuss its extension to polynomials involving shift operators.

*Conventions.* As we said above, most algorithms in this paper are written in a mixed algebraic / boolean complexity model, over an abstract field $\mathbb{K}$.

As usual, all operations $+, -, \times$ in $\mathbb{K}$ have unit cost. We will also use (sometimes implicitly) the following operations: converting $a \in \mathbb{Z}$ into its canonical image $\varphi(a) = 1 + \cdots + 1$ ($a$ times) $\in \mathbb{K}$, and conversely taking $b$ in $\mathrm{Im}(\varphi)$ and return its canonical preimage $\psi(b)$ in $\mathbb{Z}$ (if $\mathbb{K}$ has positive characteristic $p$, we take $\psi(b)$ in $[0, p)$). By convention, computing $\varphi(a)$ will take 1 operation in $\mathbb{K}$ and $O(\log(|a|))$ bit operations; computing $\psi(b)$ will take 1 operation in $\mathbb{K}$ and $O(\log(|\psi(b)|))$ bit operations.

Given $\boldsymbol{b} = (b_1, \ldots, b_n)$ in $\mathbb{K}^n$ and $\boldsymbol{d} = (d_1, \ldots, d_n)$ in $\mathbb{N}^n$, we will write $\boldsymbol{b}^{\boldsymbol{d}} = b_1^{d_1} \cdots b_n^{d_n} \in \mathbb{K}$. If $d_i \le d$ holds for all $i$, we can compute $\boldsymbol{b}^{\boldsymbol{d}}$ in $O(n \log(d))$ operations in $\mathbb{K}$ and $O(n \log(d))$ bit operations.

For $\boldsymbol{b}$ as above and $d$ in $\mathbb{N}$, we will write $\boldsymbol{b}^d = (b_1^d, \ldots, b_n^d) \in \mathbb{K}^n$; it can be computed in $O(n \log(d))$ operations in $\mathbb{K}$, and the same number of bit operations.

*Acknowledgements.* We thank the reviewers for their very helpful remarks.

## 2 A MODIFIED PRONY ALGORITHM BASED ON DERIVATIVES

We first consider the problem of interpolating sparse, commutative polynomials in variables $z_1, \ldots, z_n$. Consider a polynomial

$$f = \sum_{k=1}^{t} f_k \boldsymbol{Z}^{\boldsymbol{e}_k} \in \mathbb{K}[z_1, \ldots, z_n] \tag{4}$$

with $f_k$ in $\mathbb{K}^*$ for all $k$, where $\boldsymbol{e}_k = (e_{k,1}, \ldots, e_{k,n})$ are pairwise distinct integer vectors and where we write $\boldsymbol{Z}^{\boldsymbol{e}_k} = z_1^{e_{k,1}} \cdots z_n^{e_{k,n}}$ for all $k$. Assuming that we are given access the values of $f$ and of its derivatives at suitable points in the base field, we show how to reconstruct the sparse representation of $f$, that is, compute all exponents $\boldsymbol{e}_k$ and coefficients $f_k$.

Prony's algorithm [9], see also [3], makes this possible through the use of linear system solving, univariate polynomial root-finding and discrete logarithm extraction. In this section, we introduce a variant of this algorithm that forgoes discrete logarithms, at the expense of slightly stronger assumptions. We use this algorithm in the following sections; the underlying idea is further developed in [17].

Definition 2.1. *A point $\boldsymbol{b}$ in $\mathbb{K}^n$ is a* good point *for $f$ as in* (4) *if for all $i \neq j$ in $\{1, \ldots, t\}$ we have $\boldsymbol{b}^{\boldsymbol{e}_i} \neq \boldsymbol{b}^{\boldsymbol{e}_j}$.*

Let then $f$ be as above, and assume that the following holds:

- the characteristic of $\mathbb{K}$ is greater than $\deg_{z_i}(f)$ for all $i$,
- we are given a good point $\boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{K}^n$ for $f$,
- we are given a bound $\tau$ on the number of terms of $f$, that is, such that the inequality $t \le \tau$ holds.

Using the notation $\boldsymbol{b}^j = (b_1^j, \ldots, b_n^j)$ introduced before, define

$$a_j = f(\boldsymbol{b}^j), \quad j = 0, \ldots, 2\tau - 1,$$
$$h_{i,j} = \frac{\partial f}{\partial z_i}(\boldsymbol{b}^j), \quad i = 1, \ldots, n,\ j = 0, \ldots, \tau - 1.$$

Let us further write $v_k = \boldsymbol{b}^{\boldsymbol{e}_k} = b_1^{e_{k,1}} \cdots b_n^{e_{k,n}}$, for $k = 1, \ldots, t$. Then the values $a_j$ are given by

$$a_j = f(\boldsymbol{b}^j) = \sum_{k=1}^{t} f_k v_k^j. \tag{5}$$

For $k \ge 1$, define the $k \times k$ matrix $M_k = (a_{i+j})_{0 \le i,j < k}$. The following property of $M_k$ is well-known.

Lemma 2.2. *For $k \ge t$, $M_k$ has rank $t$.*

Proof. This follows from the following factorization of $M_k$ as a product of square matrices, valid for $k \ge t$:

$$M_k = \begin{pmatrix} 1 & 1 & \cdots & 1 & 0 & \cdots \\ v_1 & v_2 & \cdots & v_t & 0 & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ v_1^{k-1} & v_2^{k-1} & \cdots & v_t^{k-1} & 0 & \cdots \end{pmatrix} \cdot \begin{pmatrix} f_1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & f_2 & 0 & \cdots & \cdots & \cdots \\ & & \ddots & & & \\ 0 & \cdots & \cdots & f_t & 0 & \cdots \\ 0 & \cdots & \cdots & \cdots & 0 & \cdots \\ \vdots & & & & \vdots & \\ & & & & \vdots & \end{pmatrix} \begin{pmatrix} 1 & v_1 & \cdots & v_1^{k-1} \\ 1 & v_2 & \cdots & v_2^{k-1} \\ \vdots & \vdots & & \vdots \\ 1 & v_t & \cdots & v_t^{k-1} \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

□

Define the term locator polynomial $\Lambda(z)$ as

$$\Lambda(z) = \prod_{k=1}^{t} (z - v_k) = z^t + \lambda_{t-1} z^{t-1} + \cdots + \lambda_1 z + \lambda_0.$$

Suppose that $t$ is known. As in the usual Prony algorithm, we can compute the coefficients $\lambda_i$ by solving the linear system

$$M_t \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{t-1} \end{pmatrix} = \begin{pmatrix} a_t \\ a_{t+1} \\ \vdots \\ a_{2t-1} \end{pmatrix}. \tag{6}$$

Once $\Lambda$ is known, its roots give us the values $v_k$. Then, using the first $t$ evaluations of $f$, we get the following transposed Vandermonde system for the coefficients of $f$:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ v_1 & v_2 & \cdots & v_t \\ \vdots & \vdots & \ddots & \vdots \\ v_1^{t-1} & v_2^{t-1} & \cdots & v_t^{t-1} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_t \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix} \tag{7}$$

It remains to recover the exponents of $f$. We will do it using the following lemma, where we use the values of the partial derivatives of $f$, rather than relying on discrete logarithms.

Lemma 2.3. *For $i = 1, \ldots, n$ and $j = 0, \ldots, \tau - 1$, we have*

$$b_i^j h_{i,j} = \sum_{k=1}^{t} f_k e_{k,i} v_k^j.$$

Proof. As

$$f = \sum_{k=1}^{t} f_k \boldsymbol{Z}^{\boldsymbol{e}_k},$$

we have

$$z_i \frac{\partial f}{\partial z_i} = \sum_{k=1}^{t} f_k e_{k,i} \boldsymbol{Z}^{\boldsymbol{e}_k}.$$

It follows that after evaluation at $\boldsymbol{b}^j = (b_1^j, \ldots, b_n^j)$, we obtain $b_i^j h_{i,j} = \sum_{k=1}^{t} f_k e_{k,i} v_k^j$, as claimed. □

For a fixed $i$ in $1, \ldots, n$, this lemma gives the equality

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ v_1 & v_2 & \cdots & v_t \\ \vdots & \vdots & \ddots & \vdots \\ v_1^{t-1} & v_2^{t-1} & \cdots & v_t^{t-1} \end{pmatrix} \begin{pmatrix} f_1 e_{1,i} \\ f_2 e_{2,i} \\ \vdots \\ f_t e_{t,i} \end{pmatrix} = \begin{pmatrix} b_i^0 h_{i,0} \\ b_i^1 h_{i,1} \\ \vdots \\ b_i^{t-1} h_{i,t-1} \end{pmatrix}. \tag{8}$$

Thus, the coefficients $f_k$ and $f_k e_{k,i}$ are computed by solving the systems (7) and (8); the exponents $e_{k,i}$, for $k = 1, \ldots, t$ are deduced by division. Note that this procedure gives us the canonical image $\varepsilon_{k,i} = \varphi(e_{k,i})$ in $\mathbb{K}$ (using the notation introduced in the discussion at the end of the introduction). Under our assumption on the characteristic of the base field, this allows us to recover the integers $e_{k,i} = \psi(\varepsilon_{k,i})$ themselves (the notation $\psi$ as well is from the introduction).

In what follows, we assume that a root-finding algorithm is available for polynomials in $\mathbb{K}[z]$, and we write $\mathsf{R}_{\mathbb{K}}(t)$ for the expected cost of root-finding for a polynomial of degree $t$ in $\mathbb{K}[z]$ (we use expected run-time here, since over finite fields, the fastest known algorithms are Las Vegas).

Algorithm 2.4 (MPA - Modified Prony algorithm).

**Input:**

- a bound $\tau$ for the number of terms of $f$ as in (4)
- the values $a_j = f(\boldsymbol{b}^j)$, for $j = 0, \ldots, 2\tau - 1$, for some $\boldsymbol{b}$ in $\mathbb{K}^n$
- the values $h_{i,j} = \frac{\partial f}{\partial z_i}(\boldsymbol{b}^j)$, for $i = 1, \ldots, n$ and $j = 0, \ldots, \tau - 1$.

**Output:** the polynomial $f = \sum_{k=1}^{t} f_k z_1^{e_{k,1}} \cdots z_n^{e_{k,n}}$

1. Find the rank $t$ of the matrix $M_\tau$ and solve equation (6) to get the coefficients of the term locator polynomial $\Lambda(z)$.
2. Find the roots $v_1, \ldots, v_t$ of $\Lambda(z)$.
3. Find the coefficients $f_1, \ldots, f_t$ by solving the transposed Vandermonde system (7).
4. For $i = 1, \ldots, n$, solve the transposed Vandermonde system (8); let $F_{1,i}, \ldots, F_{t,i}$ be the solutions
5. For $i = 1, \ldots, n$ and $k = 1, \ldots, t$, compute $\varepsilon_{k,i} = F_{k,i}/f_k$ and $e_{k,i} = \psi(\varepsilon_{k,i}) \in \mathbb{Z}$.
6. Return $\sum_{k=1}^{t} f_k z_1^{e_{k,1}} \cdots z_n^{e_{k,n}}$.

Proposition 2.5. *Suppose that all partial degrees* $\deg_{z_i}(f)$ *satisfy* $\deg_{z_i}(f) < r$. *If* $\mathbb{K}$ *has characteristic at least* $r$ *and if* $\boldsymbol{b}$ *is a good point for* $f$, *Algorithm 2.4 returns* $f$ *using* $O^{\sim}(n\tau)$ *operations in* $\mathbb{K}$, $O^{\sim}(n\tau \log(r))$ *bit operations, and an additional cost* $\mathsf{R}_{\mathbb{K}}(\tau)$.

Proof. Since $M_\tau$ a Hankel matrix, we can find its rank $t$ using $O^{\sim}(\tau)$ operations in $\mathbb{K}$ [19, Section 4]; then, we solve Eq. (6) in time $O^{\sim}(t)$ (ibid.) By definition, root-finding to compute all $v_1, \ldots, v_t$ costs $\mathsf{R}_{\mathbb{K}}(t)$. Solving all transposed Vandermonde systems takes $O^{\sim}(nt)$ $\mathbb{K}$-operations [19]; computing all $e_{k,i}$ takes $O^{\sim}(nt)$ $\mathbb{K}$-operations and $O(nt \log(r))$ bit operations. Using the upper bound $t \le \tau$, we obtain the claimed costs. □

Prony's algorithm computes the exponents $e_{k,i}$ from $v_1, \ldots, v_t$, for which no non-trivial algorithms are known over general fields. If $\mathbb{K}$ is finite, dedicated algorithms exist, see [20] for the most recent result to date, and references therein.

Algorithm 2.4 needs $\boldsymbol{b}$ to be a good point for $f$, but unless we already know all the monomials of $f$, we don't know how to quickly verify whether it is the case. As usual, we may overcome this problem by random selection; we discuss this in a slightly more general context in the last section.

## 3 INTERPOLATION OF LINEAR DIFFERENTIAL OPERATORS

Let thus $P \in \mathbb{K}[x_1, \ldots, x_n]\langle \delta_1, \ldots, \delta_n \rangle$ be a linear operator, which we write as in the introduction as

$$P = \sum_{j=1}^{T_P} p_j \boldsymbol{X}^{\boldsymbol{d}_j} \boldsymbol{\Delta}^{\boldsymbol{r}_j} \tag{9}$$

with all coefficients $p_j$ in $\mathbb{K}^*$. Recall that we also wrote

$$P = \sum_{k=1}^{\rho} \boldsymbol{X}^{\boldsymbol{d}_k} P_{\boldsymbol{d}_k}(\delta_1, \ldots, \delta_n), \tag{10}$$

for some polynomials $P_{\boldsymbol{d}_1}, \ldots, P_{\boldsymbol{d}_\rho}$ in $\mathbb{K}[z_1, \ldots, z_n]$.

The idea of our main algorithm is to combine van der Hoeven's approach with Prony's algorithm. In this section, we give an algorithm for the sparse interpolation of such a differential operator, based on the modified version of Prony's algorithm from the previous section; this will be done by applying this algorithm to all polynomial coefficients $P_{\boldsymbol{d}_k}$ as given above. We start by defining an operation of evaluation of linear operators at points in $\mathbb{K}^n$.

Definition 3.1. *Let* $P$ *be in* $\mathbb{K}[x_1, \ldots, x_n]\langle \delta_1, \ldots, \delta_n \rangle$ *as in* (9) *and let* $\boldsymbol{b}$ *be in* $\mathbb{K}^n$. *We define the evaluation of* $P$ *at the point* $\boldsymbol{b}$ *as*

$$\begin{aligned} P(\boldsymbol{b}) &= \sum_{j=1}^{T_P} p_j \boldsymbol{b}^{\boldsymbol{r}_j} \boldsymbol{X}^{\boldsymbol{d}_j} \\ &= \sum_{k=1}^{\rho} P_{\boldsymbol{d}_k}(\boldsymbol{b}) \boldsymbol{X}^{\boldsymbol{d}_k} \in \mathbb{K}[x_1, \ldots, x_n]. \end{aligned} \tag{11}$$

The commutation rule $\delta_j x_j = x_j \delta_j + x_j$ implies that $\delta_j^i(x_j^b) = b^i x_j^b$, so Definition 3.1 of evaluation of $P$ at the point $\boldsymbol{b}$ is natural for $\boldsymbol{b} \in \mathbb{N}^n$, as it describes the application of $P$ to $x_1^{b_1} \cdots x_n^{b_n}$.

Since we want to use our modified Prony algorithm, we also need access to the values of the derivatives of the polynomials $P_{\boldsymbol{d}_k}$. This will be done through the following operation of differentiation of linear operators.

Definition 3.2. *Let* $P$ *be in* $\mathbb{K}[x_1, \ldots, x_n]\langle \delta_1, \ldots, \delta_n \rangle$ *as in* (9). *For* $i = 1, \ldots, n$, *we define the* $i$*-derivative* $\frac{\partial P}{\partial \delta_i}$ *of* $P$ *as*

$$\frac{\partial P}{\partial \delta_i} = \sum_{j=1}^{T_P} r_{j,i} p_j \boldsymbol{X}^{\boldsymbol{d}_j} \boldsymbol{\Delta}^{\boldsymbol{r}_j - \boldsymbol{I}_i},$$

*where* $\boldsymbol{I}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ *is the i-th unit vector (if* $r_{j,i} = 0$, $r_{j,i}\boldsymbol{X}^{\boldsymbol{d}_j}\boldsymbol{\Delta}^{\boldsymbol{r}_j - \boldsymbol{I}_i}$ *is zero as well).*

The following property will be useful in the next section.

Lemma 3.3. *Let* $P, Q, R$ *be in* $\mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$ *and* $i$ *be in* $1, \ldots, n$. *Then*

- *if* $P = Q + R$, *then* $\frac{\partial P}{\partial \delta_i} = \frac{\partial Q}{\partial \delta_i} + \frac{\partial R}{\partial \delta_i}$,
- *if* $P = QR$, *then* $\frac{\partial P}{\partial \delta_i} = \frac{\partial Q}{\partial \delta_i}R + Q\frac{\partial R}{\partial \delta_i}$.

Proof. Additivity is clear. To prove the second property, by additivity, it suffices to show that it holds for $Q = \boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_1}$ and $R = \boldsymbol{X}^{\boldsymbol{d}_2}\boldsymbol{\Delta}^{\boldsymbol{r}_2}$, with $\boldsymbol{d}_k = (d_{k,1}, \ldots, d_{k,n})$ and $\boldsymbol{r}_k = (r_{k,1}, \ldots, r_{k,n})$ for $k = 1, 2$. For any index $i$, $x_i$ and $\delta_i$ satisfy the relations

$$\delta_i^\ell x_i^j = x_i^j(\delta_i + j)^\ell \quad \text{for } \ell, j \in \mathbb{N}. \tag{12}$$

It follows that

$$\begin{aligned} P &= QR \\ &= \boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_1}\boldsymbol{X}^{\boldsymbol{d}_2}\boldsymbol{\Delta}^{\boldsymbol{r}_2} \\ &= \boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{X}^{\boldsymbol{d}_2}(\boldsymbol{\Delta} + \mathbf{d}_2)^{\boldsymbol{r}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_2}; \end{aligned}$$

from this, our definition gives

$$\frac{\partial P}{\partial \delta_i} = r_{1,i}\boldsymbol{X}^{\boldsymbol{d}_1+\boldsymbol{d}_2}(\boldsymbol{\Delta} + \boldsymbol{d}_2)^{\boldsymbol{r}_1-\boldsymbol{I}_i}\boldsymbol{\Delta}^{\boldsymbol{r}_2} + r_{2,i}\boldsymbol{X}^{\boldsymbol{d}_1+\boldsymbol{d}_2}(\boldsymbol{\Delta} + \boldsymbol{d}_2)^{\boldsymbol{r}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_2-\boldsymbol{I}_i}.$$

Note that if $r_{1,i}$ or $r_{2,i}$ vanishes, some exponents in this expression are negative (but the corresponding terms vanish, as they are multiplied by $r_{1,i}$, resp. $r_{2,i}$, as in Definition 3.2). Now we compute $\frac{\partial Q}{\partial \delta_i}R + Q\frac{\partial R}{\partial \delta_i}$:

$$\begin{aligned} &\frac{\partial Q}{\partial \delta_i}R + Q\frac{\partial R}{\partial \delta_i} \\ &= r_{1,i}\boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_1-\boldsymbol{I}_i}\boldsymbol{X}^{\boldsymbol{d}_2}\boldsymbol{\Delta}^{\boldsymbol{r}_2} + r_{2,i}\boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_1}\boldsymbol{X}^{\boldsymbol{d}_2}\boldsymbol{\Delta}^{\boldsymbol{r}_2-\boldsymbol{I}_i} \\ &= r_{1,i}\boldsymbol{X}^{\boldsymbol{d}_1+\boldsymbol{d}_2}(\boldsymbol{\Delta} + \boldsymbol{d}_2)^{\boldsymbol{r}_1-\boldsymbol{I}_i}\boldsymbol{\Delta}^{\boldsymbol{r}_2} + r_{2,i}\boldsymbol{X}^{\boldsymbol{d}_1+\boldsymbol{d}_2}(\boldsymbol{\Delta} + \boldsymbol{d}_2)^{\boldsymbol{r}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_2-\boldsymbol{I}_i}, \end{aligned}$$

where the second equality is due to (12). The conclusion follows. □

Starting from the expression $P = \sum_{k=1}^{\rho}\boldsymbol{X}^{\boldsymbol{d}_k}P_{\boldsymbol{d}_k}(\delta_1, \ldots, \delta_n)$ of (10), we obtain

$$\frac{\partial P}{\partial \delta_i} = \sum_{k=1}^{\rho}\boldsymbol{X}^{\boldsymbol{d}_k}\frac{\partial P_{\boldsymbol{d}_k}}{\partial z_i}(\delta_1, \ldots, \delta_n).$$

In particular, in terms of evaluations, we get

$$\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}) = \sum_{k=1}^{\rho}\frac{\partial P_{\boldsymbol{d}_k}}{\partial z_i}(\boldsymbol{b})\boldsymbol{X}^{\boldsymbol{d}_k}. \tag{13}$$

We can now present our interpolation algorithm. As input, we assume that we are given a bound $\tau$ on the $\delta$-sparsity of $P$, that is, such that all $P_{\boldsymbol{d}_k}$ have at most $\tau$ non-zero terms; the algorithm does not need $\rho$ as input, it is discovered through the procedure. The number of terms $T_P$ is bounded above by $\rho\tau$, so that the sparse representation of $P$ uses $O(\rho\tau)$ elements of $\mathbb{K}$ and $O(n\rho\tau\log(rd))$ bits.

In order to recover $P$, we interpolate all $P_{\boldsymbol{d}_k}$, using the algorithm of the previous section; in particular, we need to know values of these polynomials, as well as of their partial derivatives. Hence, we assume that we are given the values

$$P(\boldsymbol{b}^0), P(\boldsymbol{b}^1), \ldots, P(\boldsymbol{b}^{2\tau-1}),$$

as well as

$$\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^0), \frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^1), \ldots, \frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^{\tau-1}),$$

for all $i$, where we write $\boldsymbol{b}^j = (b_1^j, \ldots, b_n^j)$ as before, for some $\boldsymbol{b}$ in $\mathbb{K}^n$. Note that these "values" are polynomials in $\mathbb{K}[x_1, \ldots, x_n]$; by coefficient extraction, using (11) and (13), they give us the evaluations of the polynomials $P_{\boldsymbol{d}_k}$ and their derivatives.

Algorithm 3.4 (ILDO - Interpolation of linear differential operators).

**Input:**

- a bound $\tau$ for the $\delta$-sparsity of $P \in \mathbb{K}[x_1, \ldots, x_n]\langle\delta_1, \ldots, \delta_n\rangle$
- $2\tau$ evaluations $P(\boldsymbol{b}^j)$ given in sparse representation, for $j = 0, \ldots, 2\tau - 1$, for some $\boldsymbol{b}$ in $\mathbb{K}^n$
- $n\tau$ evaluations $\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^j)$ given in sparse representation, for $i = 1, \ldots, n$ and $j = 0, \ldots, \tau - 1$.

**Output:** the sparse representation of $P$

**Step 1:** Let $\{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_\rho\}$ be the union of the exponent sets of all $P(\boldsymbol{b}^j)$, $j = 0, \ldots, 2\tau - 1$, so that we can write

$$P(\boldsymbol{b}^j) = c_{j,1}\boldsymbol{X}^{\boldsymbol{d}_1} + \cdots + c_{j,\rho}\boldsymbol{X}^{\boldsymbol{d}_\rho}, \quad j = 0, \ldots, 2\tau - 1.$$

We will show that we can write

$$\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^j) = m_{i,j,1}\boldsymbol{X}^{\boldsymbol{d}_1} + \cdots + m_{i,j,\rho}\boldsymbol{X}^{\boldsymbol{d}_\rho}, \quad j = 0, \ldots, \tau - 1.$$

**Step 2:** Let $P_{\boldsymbol{d}_k} = \mathrm{MPA}(\tau, [c_{j,k} \mid j = 0, \ldots, 2\tau - 1], [m_{i,j,k} \mid i = 1, \ldots, n, j = 0, \ldots, \tau - 1])$, for $k = 1, \ldots, \rho$.

**Step 3:** Return $\sum_{k=1}^{\rho}\boldsymbol{X}^{\boldsymbol{d}_k}P_{\boldsymbol{d}_k}(\delta_1, \ldots, \delta_n)$.

Proposition 3.5. *Suppose that* $P$ *has bidegree less than* $(d, r)$. *If* $\mathbb{K}$ *has characteristic at least* $r$ *and if* $\boldsymbol{b}$ *is a good point for all* $P_{\boldsymbol{d}_k}$, *Algorithm 3.4 returns* $P$ *using* $O^{\sim}(n\rho\tau)$ *operations in* $\mathbb{K}$, $O^{\sim}(n\tau\rho\log(rd))$ *bit operations, and an additional cost* $\rho\mathrm{R}_{\mathbb{K}}(\tau)$.

Proof. Write (temporarily) $P = \sum_{k=1}^{\sigma}\boldsymbol{X}^{\boldsymbol{e}_k}P_{\boldsymbol{e}_k}(\delta_1, \ldots, \delta_n)$, so that for $j = 0, \ldots, 2\tau - 1$, $P(\boldsymbol{b}^j) = \sum_{k=1}^{\sigma}P_{\boldsymbol{e}_k}(\boldsymbol{b}^j)\boldsymbol{X}^{\boldsymbol{e}_k}$. We now prove that the union of exponent sets of all $P(\boldsymbol{b}^j)$ is precisely $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_\sigma\}$, so that $\sigma$ is indeed equal to the quantity $\rho$ computed in the algorithm; this will also establish the claim made at Step 1 in the pseudocode on the derivatives of $P$. The only non-trivial direction is to observe that every $\boldsymbol{e}_k$ shows up in the exponent set of at least one $P(\boldsymbol{b}^j)$, that is, that not all $P_{\boldsymbol{e}_k}(\boldsymbol{b}^j)$ can vanish, for $j = 0, \ldots, 2\tau - 1$. Since any $P_{\boldsymbol{e}_k}$ has at most $\tau$ terms, this follows e.g. from the correctness of Prony's algorithm.

From this, correctness of the algorithm follows, so we can focus on its runtime. By assumption, all inputs $P(\boldsymbol{b}^j)$ and $\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^j)$ have degree less than $d$ and at most $\rho$ terms. In particular, at Step 1, taking the union of the exponent sets of all $P(\boldsymbol{b}^j)$ takes $O^{\sim}(n\tau\rho\log(d))$ bit operations.

By Proposition 2.5, each call to algorithm MPA takes $O^{\sim}(n\tau)$ operations in $\mathbb{K}$, $O^{\sim}(n\tau\log(r))$ bit operations, as well as an extra cost $\mathrm{R}_{\mathbb{K}}(\tau)$; the conclusion follows. □

# 4 SPARSE MULTIPLICATION OF LINEAR DIFFERENTIAL OPERATORS

We can now present our sparse multiplication algorithm for linear differential operators. Given

$$A = \sum_{i=1}^{T_A} a_i \boldsymbol{X}^{\boldsymbol{d}_i} \boldsymbol{\Delta}^{\boldsymbol{r}_i} \quad \text{and} \quad B = \sum_{i=1}^{T_B} b_i \boldsymbol{X}^{\boldsymbol{e}_i} \boldsymbol{\Delta}^{\boldsymbol{s}_i} \tag{14}$$

where all $a_i, b_i$ are in $\mathbb{K}^*$, and all $\boldsymbol{d}_i, \boldsymbol{r}_i, \boldsymbol{e}_i, \boldsymbol{s}_i$ are vectors in $\mathbb{N}^n$, the aim is to compute the product $P = AB$ in sparse representation.

This is done by evaluation and interpolation at the powers of a point $\boldsymbol{b} \in \mathbb{K}^n$. The interpolation algorithm was given in the previous section; what remains is to determine the cost of computing the required "values" (recall that these are actually polynomials in $\mathbb{K}[x_1, \ldots, x_n]$). We start by computing the values of $P = AB$.

Proposition 4.1. *Assume $A, B$ are given as in (14), and that $P = AB$ has bidegree less than $(d, r)$. Then, for $\boldsymbol{b}$ in $\mathbb{K}^n$ and $\tau$ in $\mathbb{N}_{>0}$, we can compute $P(\boldsymbol{b}^k)$, for $k = 0, \ldots, 2\tau - 1$ using $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$ and $O^\sim(n\tau T_A T_B \log(rd))$ bit operations.*

Proof. For $1 \le i \le T_A$ and $1 \le j \le T_B$, let $A_i = a_i \boldsymbol{X}^{\boldsymbol{d}_i} \boldsymbol{\Delta}^{\boldsymbol{r}_i}$ and $B_j = b_j \boldsymbol{X}^{\boldsymbol{e}_j} \boldsymbol{\Delta}^{\boldsymbol{s}_j}$. Then, for $k \ge 0$,

$$P(\boldsymbol{b}^k) = \sum_{i,j} (A_i B_j)(\boldsymbol{b}^k).$$

Computing $\boldsymbol{b}^k$, for $k = 0, \ldots, 2\tau - 1$, costs $O^\sim(n\tau)$ $\mathbb{K}$-operations. Next, we analyze the cost of computing $(A_i B_j)(\boldsymbol{b}^k)$ for fixed $i, j, k$. Due to Eq. (12), we have

$$(A_i B_j)(\boldsymbol{b}^k) = a_i b_j (\boldsymbol{b}^k)^{\boldsymbol{s}_j} (\boldsymbol{b}^k + \boldsymbol{e}_j)^{\boldsymbol{r}_i} \boldsymbol{X}^{\boldsymbol{d}_i + \boldsymbol{e}_j}.$$

Computing $(\boldsymbol{b}^k)^{\boldsymbol{s}_j} (\boldsymbol{b}^k + \boldsymbol{e}_j)^{\boldsymbol{r}_i}$ costs $O^\sim(n \log(r))$ $\mathbb{K}$-operations and $O(n \log(rd))$ bit operations. Computing the exponent $\boldsymbol{d}_i + \boldsymbol{e}_j$ costs $O(n \log(d))$ bit operations, and multiplying by $a_i b_j$ costs an extra $O(1)$ $\mathbb{K}$-operations.

Since there are $T_A T_B$ pairs $A_i, B_j$ and $O(\tau)$ exponents $k$ to consider, the total cost is $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$, as well as $O^\sim(n\tau T_A T_B \log(rd))$ bit operations. □

Our interpolation algorithm also needs to know the values of the derivatives of $P$. The following elementary lemma will be useful.

Lemma 4.2. *Let $\boldsymbol{b}$ be in $\mathbb{K}^n$ and $\boldsymbol{r} = (r_1, \ldots, r_n)$ in $\mathbb{N}^n$, with $r_i \le r$ for all $i$, for some $r$ in $\mathbb{N}$. Then we can compute all $r_i \boldsymbol{b}^{\boldsymbol{r} - \boldsymbol{I}_i}$, for $i = 1, \ldots, n$, using $O^\sim(n \log(r))$ operations in $\mathbb{K}$, and the same asymptotic number of bit operations (if $r_i = 0$, then $r_i \boldsymbol{b}^{\boldsymbol{r} - \boldsymbol{I}_i}$ is by definition 0 as well).*

Proof. Let $M_i = r_i \boldsymbol{b}^{\boldsymbol{r} - \boldsymbol{I}_i}$, for $i = 1, \ldots, n$. Without loss of generality, we can assume that $r_i > 0$ for all $i$ (if $r_i = 0$, then $M_i = 0$, and $b_i$ plays no role in the computation of the other $M_j$'s).

If there are at least two zero elements in $(b_1, \ldots, b_n)$, then all $M_i$ vanish, and we are done in this case.

Suppose that there exists one index, say $k$, for which $b_k = 0$. For $i \ne k$, $M_i = 0$. For index $k$, if $r_k > 1$, then $M_k = 0$ by definition, and we are done. Otherwise, $r_k = 1$ and $M_k = r_k \boldsymbol{b}^{\boldsymbol{r} - \boldsymbol{I}_k} = r_k b_1^{r_1} b_2^{r_2} \cdots b_{k-1}^{r_{k-1}} b_{k+1}^{r_{k+1}} \cdots b_n^{r_n}$, and it can be computed in the prescribed time.

Finally, suppose that no $b_k$ vanishes. We compute $M = \boldsymbol{b}^{\boldsymbol{r}} = b_1^{r_1} \cdots b_n^{r_n}$ in $O^\sim(n \log(r))$ operations in $\mathbb{K}$, and the same number of bit operations; then, we return all $r_i M / b_i$, $i = 1, \ldots, n$. □

We can now analyze the cost of evaluating the derivatives of $P$.

Proposition 4.3. *Assume $A, B$ are given as in (14), and that $P = AB$ has bidegree less than $(d, r)$. Then, for $\boldsymbol{b}$ in $\mathbb{K}^n$ and $\tau$ in $\mathbb{N}_{>0}$, we can compute $\frac{\partial P}{\partial \delta_i}(\boldsymbol{b}^k)$, for $i = 1, \ldots, n$ and $k = 0, \ldots, \tau - 1$ using $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$ and $O^\sim(n\tau T_A T_B \log(rd))$ bit operations.*

Proof. As for Proposition 4.1, we can just consider one term in each of $A$ and $B$, respectively; without loss of generality, we consider terms $A_1 = a \boldsymbol{X}^{\boldsymbol{d}} \boldsymbol{\Delta}^{\boldsymbol{r}}$ and $B_1 = b \boldsymbol{X}^{\boldsymbol{e}} \boldsymbol{\Delta}^{\boldsymbol{s}}$, so $A_1 B_1 = ab \boldsymbol{X}^{\boldsymbol{d}} \boldsymbol{\Delta}^{\boldsymbol{r}} \boldsymbol{X}^{\boldsymbol{e}} \boldsymbol{\Delta}^{\boldsymbol{s}}$. By Lemma 3.3, we have

$$\frac{\partial (A_1 B_1)}{\partial \delta_i} = \frac{\partial A_1}{\partial \delta_i} B_1 + A_1 \frac{\partial B_1}{\partial \delta_i}, \quad i = 1, \ldots, n.$$

For $i$ in $1, \ldots, n$, this implies

$$\frac{\partial (A_1 B_1)}{\partial \delta_i}(\boldsymbol{b}^k) = abr_i (\boldsymbol{b}^k + \boldsymbol{e})^{\boldsymbol{r} - \boldsymbol{I}_i} (\boldsymbol{b}^k)^{\boldsymbol{s}} \boldsymbol{X}^{\boldsymbol{d} + \boldsymbol{e}} + abs_i (\boldsymbol{b}^k + \boldsymbol{e})^{\boldsymbol{r}} (\boldsymbol{b}^k)^{\boldsymbol{s} - \boldsymbol{I}_i} \boldsymbol{X}^{\boldsymbol{d} + \boldsymbol{e}}.$$

Fix $k$ in $0, \ldots, \tau - 1$. Then, by Lemma 4.2, it takes $O^\sim(n \log(r))$ operations in $\mathbb{K}$, and the same number of bit operations, to compute all $r_i (\boldsymbol{b}^k + \boldsymbol{e})^{\boldsymbol{r} - \boldsymbol{I}_i}$ and $s_i (\boldsymbol{b}^k)^{\boldsymbol{s} - \boldsymbol{I}_i}$, for $i = 1, \ldots, n$, and the same hold for $(\boldsymbol{b}^k)^{\boldsymbol{s}}$ and $(\boldsymbol{b}^k + \boldsymbol{e})^{\boldsymbol{r}}$. It takes $O(n \log(d))$ bit operations to compute $\boldsymbol{d} + \boldsymbol{e}$, so overall, the cost for fixed $k$ is $O^\sim(n \log(r))$ operations in $\mathbb{K}$ and $O^\sim(n \log(rd))$ bit operations. Taking all terms in $A$ and $B$ into consideration, the conclusion follows. □

Algorithm 4.4 (Multiplication of two linear differential operators).

**Input:**

- two linear differential operators $A, B \in \mathbb{K}[x_1, \ldots, x_n]\langle \delta_1, \ldots, \delta_n \rangle$ as in (14)
- a $\delta$-sparsity bound $\tau$ for $P = AB$
- a point $\boldsymbol{b}$ in $\mathbb{K}^n$

**Output:** the sparse representation of the product $P = AB$.

**Step 1:** For $k = 0, \ldots, 2\tau - 1$, compute $(AB)(\boldsymbol{b}^k)$.
**Step 2:** For $i = 1, \ldots, n$ and $k = 0, \ldots, \tau - 1$, compute $\frac{\partial (AB)}{\partial \delta_i}(\boldsymbol{b}^k)$.
**Step 3:** Return $\mathrm{ILDO}(\tau, \boldsymbol{b}, ((AB)(\boldsymbol{b}^k)_{k=0,\ldots,2\tau-1}), (\frac{\partial (AB)}{\partial \delta_i}(\boldsymbol{b}^k)_{i=1,\ldots,n,k=0,\ldots,\tau-1}))$.

Proposition 4.5. *Suppose that $A$, resp. $B$, has $T_A$ terms, resp. $T_B$ terms, and that $P$ has bidegree less than $(d, r)$. If $\mathbb{K}$ has characteristic at least $r$ and if $\boldsymbol{b}$ is a good point for all $P_{\boldsymbol{d}_k}$, Algorithm 4.4 returns $P$ using $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$, $O^\sim(n\tau T_A T_B \log(rd))$ bit operations and an extra cost $T_A T_B \mathsf{R}_{\mathbb{K}}(\tau)$.*

Proof. Correctness of the algorithm is clear from the previous discussion; we now analyze runtime.

By Proposition 4.1, Step 1 takes $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$ and $O^\sim(n\tau T_A T_B \log(rd))$ bit operations. By Proposition 4.3, Step 2 uses $O^\sim(n\tau T_A T_B \log(r))$ operations in $\mathbb{K}$ and $O^\sim(n\tau T_A T_B \log(rd))$ bit operations as well.

Finally, by Proposition 3.5, the cost of Step 3 is $O^{\sim}(n\rho\tau)$ operations in $\mathbb{K}$, $O^{\sim}(n\tau\rho\log(rd))$ bit operations, and an extra cost $\rho\mathsf{R}_{\mathbb{K}}(\tau)$. Because of the commutation rule

$$\boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_1}\boldsymbol{X}^{\boldsymbol{d}_2}\boldsymbol{\Delta}^{\boldsymbol{r}_2} = \boldsymbol{X}^{\boldsymbol{d}_1}\boldsymbol{X}^{\boldsymbol{d}_2}(\boldsymbol{\Delta}+\mathbf{d}_2)^{\boldsymbol{r}_1}\boldsymbol{\Delta}^{\boldsymbol{r}_2},$$

we have $\rho \le T_A T_B$, so the runtime above is $O^{\sim}(n\tau T_A T_B)$ operations in $\mathbb{K}$, $O^{\sim}(n\tau T_A T_B\log(rd))$ bit operations, and an extra $T_A T_B \mathsf{R}_{\mathbb{K}}(\tau)$ to account for root-finding. □

The previous algorithm needs $\boldsymbol{b}$ to be a good point for all polynomials $P_{\boldsymbol{d}_k}$. Assuming that $\boldsymbol{b}$ is chosen at random, we quantify the probability that this be the case. Recall that we write

$$P = \sum_{k=1}^{\rho} \boldsymbol{X}^{\boldsymbol{d}_k} P_{\boldsymbol{d}_k}(\delta_1,\dots,\delta_n),$$

where each $P_{\boldsymbol{d}_k}$ is a polynomial of the form

$$P_{\boldsymbol{d}_k} = \sum_{1\le i\le t_k} p_{i,k}\boldsymbol{Z}^{\boldsymbol{r}_{i,k}} \in \mathbb{K}[z_1,\dots,z_n],$$

for some integer $t_k$, non-zero coefficients $p_{i,k}$ and pairwise distinct exponents $\boldsymbol{r}_{i,k}$. Then, a point $\boldsymbol{b}\in\mathbb{K}^n$ satisfies our condition if it cancels none of the polynomials

$$\Gamma_k = \prod_{1\le i<j\le t_k} (\boldsymbol{Z}^{\boldsymbol{r}_{i,k}} - \boldsymbol{Z}^{\boldsymbol{r}_{j,k}}) \in \mathbb{K}[z_1,\dots,z_n].$$

Proposition 4.6. *Suppose that $P$ has bidegree less than $(d,r)$ and $T_P$ non-zero terms. For a subset $S$ of $\mathbb{K}$, the probability that $\boldsymbol{b}$ chosen uniformly at random in $S^n$ is a good point for all $P_{\boldsymbol{d}_k}$ is at least*

$$1 - \frac{nrT_P(T_P-1)}{2|S|}.$$

Proof. For a fixed $k$, since $\Gamma_k$ has degree less than $nrt_k(t_k-1)/2$, by the DeMillo-Lipton-Schwartz-Zippel lemma, there exist no more than $nrt_k(t_k-1)|S|^{n-1}/2$ $n$-uples $\boldsymbol{b}$ that cancel it in $S^n$.

Since $t_1+\cdots+t_\rho = T_P$, the number of $n$-uples $\boldsymbol{b}$ that cancel one of $\Gamma_1,\dots,\Gamma_k$ in $S^n$ is at most $nrT_P(T_P-1)|S|^{n-1}/2$. □

Note that if $\mathbb{K}$ is a finite field, Proposition 8 in [24] gives a slightly sharper bound $1-\frac{rT_P(T_P-1)}{2|S|}$, if we take $S=\mathbb{K}^*$ (so $|S|=|\mathbb{K}|-1$).

Still working with $\mathbb{K}$ finite, say $\mathbb{K}=\mathbb{F}_q$, we can now give the proof of Theorem 1.1. In view of the previous remark, to ensure a success rate of at least $\frac{3}{4}$, we want our base field to have cardinality at least $2rT_P(T_P-1)+1$.

If $q \ge 2rT_P(T_P-1)+1$, then we randomly choose $\boldsymbol{b}$ from $\mathbb{F}_q^{*\boldsymbol{n}}$; since we assume that we can get a random bit in time $O(1)$, this costs $O^{\sim}(n\log(q))$ bit operations. We can then invoke Proposition 4.5, taking into account that all operations in $\mathbb{K}=\mathbb{F}_q$ take $O^{\sim}(n\log(q))$ bit operations. Root-finding in degree $\tau$ in $\mathbb{F}_q$ takes an expected $O^{\sim}(\tau\log^2(q))$ bit operations, so the overall runtime becomes an expected $O^{\sim}(n\tau T_A T_B\log(rd)\log(q)+\tau T_A T_B\log^2(q))$, as claimed.

If $q < rT_P(T_P-1)+1$, we will work in suitable extension $\mathbb{K}=\mathbb{F}_q\subset\mathbb{F}_{q^u}$, where the latter field is represented as $\mathbb{F}_q[x]/\langle\Phi(x)\rangle$, for a degree-$u$ irreducible polynomial $\Phi$ over $\mathbb{F}_q$. With this representation, arithmetic operations in $\mathbb{F}_{q^u}$ can be done in $O^{\sim}(u)$ arithmetic operations in $\mathbb{K}$, and thus in $O^{\sim}(u\log(q))$ bit operations. To guarantee $q^u \ge 2rT_P(T_P-1)+1$, we choose

$$u = \lceil \frac{\log(2rT_P(T_P-1)+1)}{\log(q)} \rceil \in O(\frac{\log(rT_P)}{\log(q)}); \tag{15}$$

note that the big-O estimate holds precisely because of our assumption $q < rT_P(T_P-1)+1$.

In [23], Shoup proved that finding an irreducible polynomial $\Phi$ with degree $u$ over $\mathbb{F}_q$ costs an expected $O^{\sim}(u^2\log(q)+u\log^2(q))$ bit operations. In view of (15), this is $O^{\sim}(\log^2(rT_P)+\log(rT_P)\log(q))$. Note that when the characteristic of $\mathbb{F}_q$ is fixed, we may use Shoup's deterministic algorithm [22] instead.

Overall, the runtime of the algorithm becomes an expected $O^{\sim}(n\tau T_A T_B\log(rd)\log(q')+\tau T_A T_B\log^2(q'))$ bit operations, with $\log(q')=u\log(q)\in O(\log(rT_P))$. Since $T_P\le\tau T_A T_B$, the cost is $O^{\sim}(n\tau T_A T_B\log(rd)^2\log^2(q))$ bit operations. The proof is complete.

Remark 4.7. *Consider the ring $\mathbb{K}[x_1,\dots,x_n]\langle S_1,\dots,S_n\rangle$, with $S_i x_i = (x_i+1)S_i$; here, $S_i$ stands for the shift operator*

$$f(x_1,\dots,x_n)\mapsto f(x_1,\dots,x_{i-1},x_i+1,x_{i+1},\dots,x_n).$$

*Inspired by a remark due to van der Hoeven in [12], we may define the "adjoint" evaluation of $\boldsymbol{X}^{\boldsymbol{d}}\boldsymbol{S}^{\boldsymbol{r}}$ at $\boldsymbol{b}$ as $\boldsymbol{b}^{\boldsymbol{d}}\boldsymbol{S}^{\boldsymbol{r}}$, where $\boldsymbol{S}=(S_1,\dots,S_n)$ and $\boldsymbol{b}$ is in $\mathbb{K}^n$ (compare with the evaluation of $\boldsymbol{X}^{\boldsymbol{d}}\boldsymbol{\Delta}^{\boldsymbol{r}}$ at $\boldsymbol{b}$ being $\boldsymbol{b}^{\boldsymbol{r}}\boldsymbol{X}^{\boldsymbol{d}}$). This analogy may be exploited in a straightforward way to adapt our algorithm to this setting.*

## REFERENCES


[1] Andrew Arnold and Daniel S. Roche. 2015. Output-Sensitive Algorithms for Sumset and Sparse Polynomial Multiplication. In *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2015, Bath, United Kingdom, July 06-09, 2015*. ACM, 29–36.

[2] Martin E. Avendano, Teresa Krick, and Ariel Pacetti. 2006. Newton-Hensel Interpolation Lifting. *Foundations of Computational Mathematics* 6, 1 (2006), 82–120.

[3] Michael Ben-Or and Prasoon Tiwari. 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 301–309.

[4] Alexandre Benoit, Alin Bostan, and Joris van der Hoeven. 2012. Quasi-optimal multiplication of linear differential operators. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE, 524–530.

[5] Alin Bostan, Frédéric Chyzak, and Nicolas Le Roux. 2008. Products of ordinary differential operators by evaluation and interpolation. In *Symbolic and Algebraic Computation, International Symposium, ISSAC 2008, Linz/Hagenberg, Austria, July 20-23, 2008, Proceedings*. ACM, 23–30.

[6] David G. Cantor and Erich Kaltofen. 1991. On Fast Multiplication of Polynomials over Arbitrary Algebras. *Acta Informatica* 28, 7 (1991), 693–701.

[7] Xavier Caruso and Jérémy Le Borgne. 2017. Fast multiplication for skew polynomials. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2017, Kaiserslautern, Germany, July 25-28, 2017*. ACM, 77–84.

[8] Don Coppersmith and Shmuel Winograd. 1990. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation* 9, 3 (1990), 251–280.

[9] Marie Riche de Prony. 1795. Essai expérimental et analytique sur les lois de la Dilatabilité des fluides élastiques et sur celles de la Force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures. *Journal de l'École polytechnique* (1795), 24–75.

[10] Franois Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*. ACM Press, 296–303.

[11] Mark Giesbrecht, Qiao-Long Huang, and Éric Schost. 2020. Sparse multiplication for skew polynomials. In *ISSAC '20: International Symposium on Symbolic and Algebraic Computation, Kalamata, Greece, July 20-23, 2020*. ACM, 194–201.

[12] Joris van der Hoeven. 2002. FFT-like multiplication of linear differential operators. *Journal of Symbolic Computation* 33, 1 (2002), 123–127.

[13] Joris van der Hoeven and Grégoire Lecerf. 2012. On the Complexity of Multivariate Blockwise Polynomial Multiplication. In *International Symposium on*

*Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012*. ACM, 211–218.

[14] Joris van der Hoeven and Grégoire Lecerf. 2015. Sparse Polynomial Interpolation in Practice. *ACM Commun. Comput. Algebra* 48, 3/4 (2015), 187–191.

[15] Joris van der Hoeven and Michael Monagan. 2020. Implementing the Tangent Graeffe Root Finding Method. In *ICMS 2020*. Springer International Publishing, 482–492.

[16] Qiao-Long Huang. 2019. Sparse Polynomial Interpolation over Fields with Large or Zero Characteristic. In *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, ISSAC 2019, Beijing, China, July 15-18, 2019*. ACM, 219–226.

[17] Qiao-Long Huang. 2020. Sparse Polynomial Interpolation Based on Derivative. https://arxiv.org/abs/2002.03708

[18] Erich Kaltofen and Wen-shin Lee. 2003. Early termination in sparse interpolation algorithms. *J. Symb. Comput.* 36, 3-4 (2003), 365–400.

[19] Erich Kaltofen and Lakshman Yagati. 1988. Improved sparse multivariate polynomial interpolation algorithms. In *Symbolic and Algebraic Computation, International Symposium ISSAC'88, Rome, Italy, July 4-8, 1988, Proceedings*. Springer, 467–474.

[20] Thorsten Kleinjung and Benjamin Wesolowski. 2019. Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic. arXiv:1906.10668 [math.NT]

[21] Daniel S. Roche. 2018. What Can (and Can't) we Do with Sparse Polynomials?. In *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*. ACM.

[22] Victor Shoup. 1990. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.* 54 (1990), 435–447.

[23] Victor Shoup. 1994. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation* 17, 5 (1994), 371–391.

[24] Richard Zippel. 1990. Interpolating polynomials from their values. *Journal of Symbolic Computation* 9, 3 (1990), 375–403.