Multi-point evaluation in higher dimensions

Joris van der Hoeven[*]

Laboratoire d'informatique
UMR 7161 CNRS
École polytechnique
91128 Palaiseau Cedex
France

Email: `vdhoeven@lix.polytechnique.fr`
Web: `http://www.lix.polytechnique.fr/~vdhoeven`

Éric Schost[†]

Computer Science Department
The University of Western Ontario
London, Ontario
Canada

Email: `eschost@uwo.ca`
Web: `http://www.csd.uwo.ca/~eschost`

December 5, 2012

In this paper, we propose efficient new algorithms for multi-dimensional multi-point evaluation and interpolation on certain subsets of so called tensor product grids. These point-sets naturally occur in the design of efficient multiplication algorithms for finite-dimensional $\mathcal{C}$-algebras of the form $\mathcal{A} = \mathcal{C}[x_1, \ldots, x_n]/I$, where I is generated by monomials of the form $x_1^{i_1} \cdots x_n^{i_n}$; one particularly important example is the algebra of truncated power series $\mathcal{C}[x_1, \ldots, x_n]/(x_1, \ldots, x_n)^d$. Similarly to what is known for multi-point evaluation and interpolation in the univariate case, our algorithms have quasi-linear time complexity. As a known consequence [Sch05], we obtain fast multiplication algorithms for algebras $\mathcal{A}$ of the above form.

Keywords: multi-point evaluation, multi-point interpolation, algorithm, complexity, power series multiplication

A.M.S. subject classification: 12Y05, 68W30, 68W40, 13P10, 65F99

1. Introduction

Overview. The purpose of this paper is to give fast algorithms for some polynomial evalua-

*. This work has been partly supported by the French ANR-09-JCJC-0098-01 MaGiX project, and by the Digiteo 2009-36HD grant of the Région Ile-de-France.

†. This work has also been supported by NSERC and the Canada Research Chairs program.

tion and interpolation problems in several variables; as an application, we improve algorithms for multiplying dense multivariate polynomials and multivariate power series.

The complexity of our algorithms will be measured by counting base field operations: we do not consider numerical issues (they may anyway be irrelevant, if e.g. our base field is a finite field), and do not discuss the choice of data structures or index manipulation issues.

From the complexity point of view, evaluation and interpolation are rather well understood for polynomials in one variable: algorithms of quasi-linear complexity are known to evaluate a polynomial of degree less than d at d points, and conversely to interpolate it. The best known algorithms [BM74] run in time $O(d \log^2 d \log\log d)$, and the main remaining question is to close the gap between this and an optimal $O(d)$, if at all possible.

In several variables, the questions are substantially harder, due to the variety of monomial bases and evaluation sets one may consider; no quasi-linear time algorithm is known in general. In this paper, following the terminology of [Sau04], we consider evaluation points that are subgrids of *tensor product grids*. We prove that for some suitable monomial bases, evaluation and interpolation can both be done in time $O(n\,|I|\log^2|I|\log\log|I|)$, where n is the number of variables and $|I|$ is the size of the evaluation set (and of the monomial basis we consider). Remark that this result directly generalizes the univariate case. In many cases, n is logarithmic in $|I|$; then, our result is optimal, up to logarithmic factors.

Moreover, for specific types of evaluation points, such as roots of unity or points in a geometric progression, even faster algorithms can be used in the univariate case, of time complexity $O(d \log d \log \log d)$. These algorithms will also be generalized to the multivariate case and result in evaluation and interpolation algorithms of time complexity $O(|I|\; n \log d \log \log d)$, where d is the maximal partial degree. In particular, given two dense multivariate polynomials in n variables of total degree $<d/2$ can be multiplied in time $O\big(\binom{n+d-1}{n}\, n \log d \log \log d\big)$. To the best of our knowledge, this is the best currently available complexity bound for this problem. We also expect the new algorithms to be efficient in practice, although we have not implemented them yet.

Problem statement. In what follows, $I \subseteq \mathbb{N}^n$ is a finite initial segment for the partial ordering on $\mathbb{N}^n$: this means that if $\boldsymbol{i} \leqslant \boldsymbol{i}'$ and $\boldsymbol{i}' \in I$, then $\boldsymbol{i} \in I$. For instance, one may think of I as the set of standard monomials modulo a 0-dimensional ideal, for a given monomial ordering. Figure 1 shows such a set (black dots), as well as the minimal elements of $\mathbb{N}^n \setminus I$ (green squares).

As a very particular example, for positive integers $d_1, \ldots, d_n$, let $I_{d_1,\ldots,d_n}$ denote the set $\{0, \ldots, d_1 - 1\} \times \cdots \times \{0, \ldots, d_n - 1\}$; this is an n-dimensional grid.

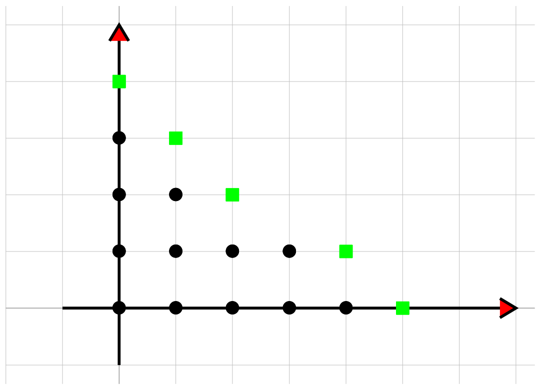

Figure 1. An initial segment of cardinality 12 in $\mathbb{N}^2$

The set I will be used as an index set for both the evaluation points and the monomial basis. Let $\mathcal{C}$ be our base field and let $d_1, \ldots, d_n$ be such that $I \subseteq I_{d_1, \ldots, d_n}$. For $k \in \{1, \ldots, n\}$, assume that we are given pairwise distinct elements $v_k = (v_{k,0}, \ldots, v_{k,d_k-1}) \in \mathcal{C}^{d_k}$; we will denote by v the collection $(v_1, \ldots, v_n)$. To $\boldsymbol{i} = (i_1, \ldots, i_n) \in I$ we associate the point $\boldsymbol{\alpha}_{\boldsymbol{i},v} = (v_{1,i_1}, \ldots, v_{n,i_n}) \in \mathcal{C}^n$ and we let $V(I, v) = \{\boldsymbol{\alpha}_{\boldsymbol{i},v} : \boldsymbol{i} \in I\}$: this will be our set of evaluation points. Remark that $V(I, v)$ is contained in the "tensor product" grid

$$(v_{1,0}, \ldots, v_{1,d_1-1}) \times \cdots \times (v_{n,0}, \ldots, v_{n,d_n-1}).$$

For instance, if $v_{k,i} = i$ for all (k, i), then $\boldsymbol{\alpha}_{\boldsymbol{i},v} = \boldsymbol{i}$ and $V(I, v) = I$.

Let further $\mathcal{C}[\boldsymbol{x}] = \mathcal{C}[x_1, \ldots, x_n]$ be the polynomial ring in n variables over $\mathcal{C}$; for $\boldsymbol{i} = (i_1, \ldots, i_n) \in \mathbb{N}^n$, we write $\boldsymbol{x}^{\boldsymbol{i}} = x_1^{i_1} \cdots x_n^{i_n}$. Then, $\mathcal{C}[\boldsymbol{x}]_I$ denotes the $\mathcal{C}$-vector space of polynomials $P = \sum_{\boldsymbol{i} \in I} p_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}} \in \mathcal{C}[\boldsymbol{x}]$ with support in I. On the example of Figure 1, $\mathcal{C}[\boldsymbol{x}]_I$ admits the monomial basis

$$1, x_2, x_2^2, x_2^3, \quad x_1, x_1 x_2, x_1 x_2^2, \quad x_1^2, x_1^2 x_2, \quad x_1^3, x_1^3 x_2, \quad x_1^4.$$

Given a polynomial $P \in \mathcal{C}[\boldsymbol{x}]_I$, written on the monomial basis, our problem of multidimensional multi-point evaluation is the computation of the vector $\{P(\boldsymbol{\alpha}_{\boldsymbol{i},v}) : \boldsymbol{i} \in I\} \in \mathcal{C}^I$.

Both the domain $\mathcal{C}[\boldsymbol{x}]_I$ and the codomain $\mathcal{C}^I$ of the evaluation map are $\mathcal{C}$-vector spaces of dimension $|I|$, so it makes sense to ask whether this map is invertible. Indeed, let $\mathfrak{I}(I, v) \subseteq \mathcal{C}[\boldsymbol{x}]$ be the defining ideal of $V(I, v)$. A result going back to Macaulay (see [Mor03] for a proof) shows that the monomials $\{\boldsymbol{x}^{\boldsymbol{i}} : \boldsymbol{i} \in I\}$ form a monomial basis of $\mathcal{C}[\boldsymbol{x}]/\mathfrak{I}(I, v)$. As a consequence, the former evaluation map is invertible; the inverse problem is an instance of multivariate interpolation.

Previous work. The purpose of this paper is to give complexity results for the evaluation and interpolation problems described above. We found no previous references dedicated to the evaluation problem (a naive solution obviously takes quadratic time). As to our form of interpolation, an early reference is [Wer80], with a focus on the bivariate case; the question has been the subject of several subsequent works, and one finds a comprehensive treatment in [Sau04]. However, the algorithms mentioned previously do not have quasi-linear complexity.

To obtain a quasi-linear result, we rely on the fast univariate algorithms of [BM74]. In the special case where I is the grid $I_{d_1,\ldots,d_n}$, Pan [Pan94] solves the multivariate problem by applying a "tensored" form of the univariate algorithms, evaluating or interpolating one variable after the other. The key contribution of our paper is the use of a multivariate Newton basis, combined with fast change of basis algorithms between the Newton basis and the monomial basis; this will allow us to follow an approach similar to Pan's in our more general situation. The Newton basis was already used in many previous works on our interpolation problem [Wer80, Müh88], accompanied by divided differences computations: we avoid divided differences, as they lead to quadratic time algorithms.

The results in this paper have a direct application to multivariate power series multiplication. Let I be as above, and let $\mathfrak{m}$ be the monomial ideal generated by $\{\boldsymbol{x}^{\boldsymbol{i}} : \boldsymbol{i} \notin I\}$; equivalently, $\mathfrak{m}$ is generated by all minimal elements of $\mathbb{N}^n \setminus I$. Then, one is interested the complexity of multiplication modulo $\mathfrak{m}$, that is, in $\mathcal{C}[\boldsymbol{x}]/\mathfrak{m}$. Suitable choices of I lead to *total degree* truncation (take $I = \{(i_1, \ldots, i_n) : i_1 + \cdots + i_n < d\}$, so $\mathfrak{m} = \langle x_1, \ldots, x_n \rangle^d$), which is used in many forms of Newton-Hensel lifting, or *partial degree* truncation (take $I = \{(i_1, \ldots, i_n) : i_1 < d_1, \ldots, i_n < d_n\}$, so $\mathfrak{m} = \langle x_1^{d_1}, \ldots, x_n^{d_n} \rangle$).

There is no known algorithm with quasi-linear cost for this question in general. Inspired by the sparse multiplication algorithm of [CKL89], Lecerf and Schost gave such an algorithm for total degree truncation [LS03]. It was extended to weighted total degree in [vdH02] and further improved from the bit-complexity point of view in [vdHL09]. Further speed-ups are possible in small dimensions, when using the Truncated Fourier Transform or TFT [vdH04, vdH05]. For more general truncation patterns, Schost [Sch05] introduced an algorithm based on deformation techniques that uses evaluation and interpolation of the form described in this paper. At the time of writing [Sch05], no efficient algorithm was known for evaluation and interpolation; the present paper fills this gap and completes the results of [Sch05].

Conventions. In all that follows, we let $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ denote a multiplication time function, in the sense that univariate polynomials of degree less than d can be multiplied in $\mathsf{M}(d)$ operations in $\mathcal{C}$. As in [GG03], we impose the condition that $\mathsf{M}(d)/d$ is an increasing function (and freely use all consequences of this assumption), and we note that M can be taken in $O(d \log d \log \log d)$ using the algorithm of [CK91].

We will use big-Oh notation for expressions that depend on an unbounded number of variables (e.g., for Lemma 6 below). In such cases, the notation $f(d_1, \ldots, d_n) = O(g(d_1, \ldots, d_n))$ means that there exists a universal constant λ such that for all n and all d_1, …, d_n, the inequality $f(d_1, \ldots, d_n) \leqslant \lambda\, g(d_1, \ldots, d_n)$ holds.

2. UNIVARIATE ALGORITHMS

This section describes some mostly classical algorithms for univariate polynomials over $\mathcal{C}$. We denote by $\mathcal{C}[x]_d$ the set of univariate polynomials of degree less than d. Given pairwise distinct points $v = v_0, \ldots, v_{d-1}$ in $\mathcal{C}$, we write $N_{i,v}(x) = (x - v_0) \cdots (x - v_{i-1})$ for $0 \leqslant i \leqslant d$. The polynomials $N_{0,v}, \ldots, N_{d-1,v}$ are called the *Newton basis* associated to v; they form a $\mathcal{C}$-basis of $\mathcal{C}[x]_d$. For instance, for $v_i = i$, we have $N_{i,v}(x) = x\,(x-1) \cdots (x - (i-1))$.

Because we will have to switch frequently between the monomial and the Newton bases, it will be convenient to use the notation $N_{i,v,\varepsilon}(x)$, for $\varepsilon \in \{0,1\}$, with

$$\begin{aligned} N_{i,v,0}(x) &= x^i \\ N_{i,v,1}(x) &= N_{i,v}(x) = (x-v_0)\cdots(x-v_{i-1}). \end{aligned}$$

We write $P \dashv (N_{i,v,\varepsilon})_{i<d}$ to indicate that a polynomial $P \in \mathcal{C}[x]_d$ is written on the basis $(N_{i,v,\varepsilon})_{i<d}$; remember that when no value ε is mentioned in subscript, we are working in the Newton basis.

2.1. General results

The following classical lemma [BP94, Ex. 15 p. 67] gives complexity estimates for conversion between these bases.

Lemma 1. *For $\varepsilon \in \{0,1\}$, given $P \dashv (N_{i,v,\varepsilon})_{i<d}$, one can compute $P \dashv (N_{i,v,1-\varepsilon})_{i<d}$ in time $O(\mathsf{M}(d)\log d)$.*

Evaluation and interpolation in the monomial basis can be done in time $O(\mathsf{M}(d)\log d)$, by the algorithms of [BM74]; combining this to the previous lemma, we obtain a similar estimate for evaluation and interpolation with respect to the Newton basis.

Lemma 2. *For $\varepsilon \in \{0,1\}$, given $P \dashv (N_{i,v,\varepsilon})_{i<d}$, one can compute $P(v_i)_{0\leqslant i<d}$, and conversely recover $P \dashv (N_{i,v,\varepsilon})_{i<d}$ from its values $P(v_i)_{0\leqslant i<d}$, in time $O(\mathsf{M}(d)\log d)$.*

If the points $v_0,\ldots,v_{d-1}$ are in geometric progression, we may remove a factor $\log d$ in all estimates. Indeed, under these assumptions, the conversions of Lemma 1 and the evaluation or interpolation of Lemma 2 take time $O(\mathsf{M}(d))$ [BS05]. The following subsection studies in detail another particular case, TFT (Truncated Fourier Transform) points. In this case we may also remove the factor $\log d$ in all estimates, but the constant factor is even better than for points in geometric progression.

2.2. TFT points

In this subsection, we are going to assume that $\mathcal{C}$ contains suitable roots of unity, and prove refined complexity bounds for such points.

Let d be as above, let $q = \lceil \log_2 d \rceil$ be the smallest integer such that $d \leqslant 2^q$ and let us suppose that $\mathcal{C}$ contains a primitive 2^q-th root of unity ζ. The TFT points are $v_i = \zeta^{[i]_q}$, for $i = 0,\ldots,d-1$, where $[i]_q$ is the binary q-bits mirror of i. In other words, they form an initial segment of length d for the sequence of 2^q-th roots of unity written in the bit-reverse order; when for instance $d=3$ and $q=2$, these points are $1, -1, \sqrt{-1}$. In this subsection, these points are fixed, so we drop the subscript $_v$ in our notation.

It is known [vdH04, vdH05] that in the monomial basis, evaluation and interpolation at the TFT points can be done in time $O(d\log d)$. Precisely, both operations can be done using $d\,q + 2^q$ shifted additions and subtractions and $\lceil (d\,q + 2^q)/2 \rceil$ multiplications by powers of ζ. Here we recall that a shifted addition (resp. subtraction) is a classical addition (resp. subtraction) where any of the inputs may be premultiplied by 2 or $1/2$ (e.g., $a \pm 2\,b$). In the most interesting case $d \simeq 2^q/2$, the TFT roughly saves a factor of 2 over the classical FFT; this makes it a very useful tool for e.g. polynomial multiplication.

We will show here that similar results hold for the conversion between the monomial and Newton bases. First, we give the basis of the algorithm by assuming that $d = 2^q$, so that $\deg(P) = 2^q - 1$. Such a polynomial can be written in the monomial, resp. Newton basis, as

$$P = \sum_{i=0}^{2^q-1} p_{i,0}\, x^i = \sum_{i=0}^{2^q-1} p_{i,1} \prod_{j=0}^{i-1} \left(x - \zeta^{[j]_q}\right).$$

For $k = 0, \ldots, q$, let us introduce the polynomials $P_0^{(k)}, \ldots, P_{2^{q-k}-1}^{(k)}$, all of degree less than 2^k, such that

$$P = \sum_{i=0}^{2^{q-k}-1} P_i^{(k)} \prod_{j=0}^{i-1} \left(x^{2^k} - \zeta^{[j]_q}\right). \tag{1}$$

Thus, $P_i^{(0)} = p_{i,1}$ for $k = 0$ and $i = 0, \ldots, 2^q - 1$, whereas $P_0^{(q)} = P$ for $k = q$.

LEMMA 3. *For $k = 0, \ldots, q-1$ and $i = 0, \ldots, 2^{q-k-1} - 1$, we have*

$$P_i^{(k+1)} = P_{2i}^{(k)} + P_{2i+1}^{(k)} \left(x^{2^k} - \zeta^{[2i]_q}\right) = \left(P_{2i}^{(k)} - \zeta^{[2i]_q} P_{2i+1}^{(k)}\right) + x^{2^k} P_{2i+1}^{(k)}.$$

Proof. This follows by grouping the terms of indices $2i$ and $2i+1$ in (1), and by noticing that for all $i < 2^{q-k-1}$, the following equality holds:

$$\prod_{j=0}^{2i-1} \left(x^{2^k} - \zeta^{[j]_q}\right) = \prod_{j=0}^{i-1} \left(x^{2^{k+1}} - \zeta^{[j]_q}\right).$$

Indeed, for all even $j < 2^q$, $[j+1]_q = -[j]_q$; thus, the left-hand side is the product of all $x^{2^{k+1}} - \zeta^{2[j]_q}$, for j (even) $= 0, 2, \ldots, 2i-2$. The claim follows by writing $j = 2j'$, observing that $2[2j']_q = [j']_q$. □

The previous lemma implies an algorithm of complexity $O(d \log d)$ that takes as input the coefficients $p_{0,1}, \ldots, p_{2^q-1,1}$ on the Newton basis, and outputs P on the monomial basis. It suffices to compute all polynomials $P_i^{(k+1)}$ (on the monomial basis) by means of the recursive formula. Computing $P_i^{(k+1)}$ from $P_{2i}^{(k)}$ and $P_{2i+1}^{(k)}$ takes 2^k additions and 2^k multiplications by powers of ζ, so going from index k to $k+1$ takes a total of $2^q - 1$ additions and 2^{q-1} multiplications. The inverse conversion takes time $O(d \log d)$ as well, since knowing $P_i^{(k+1)}$, we can recover first $P_{2i+1}^{(k)}$ for free as the high-degree terms of $P_i^{(k+1)}$, then $P_{2i}^{(k)}$ using 2^k additions and 2^k multiplications.

The conversion algorithm from the Newton to the monomial basis can be depicted as follows, in the case $d = 16$, $q = 4$. The flow of the algorithm goes down, from $k = 0$ to $k = 4$; the kth row contains the 16 coefficients of the polynomials $P_0^{(k)}, \ldots, P_{2^{4-k}-1}^{(k)}$ (on the monomial basis), in that order, and each oblique line corresponds to a multiplication by a root of unity. The algorithm does only "half-butterflies", compared to the FFT algorithm.

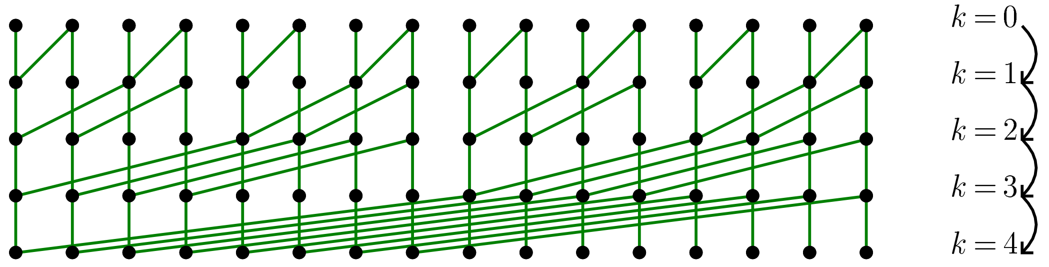

Figure 2. Schematic representation of the conversion Newton to monomial for $d=16$

If we assume that $d<2^q$, we will be able to avoid useless computations, by keeping track of the zero coefficients in the polynomials $P_i^{(k)}$. The next figure shows the situation for $d=11$; this is similar to what happens in van der Hoeven's TFT algorithm, but much simpler (here, at each level, we can easily locate the zero coefficients).

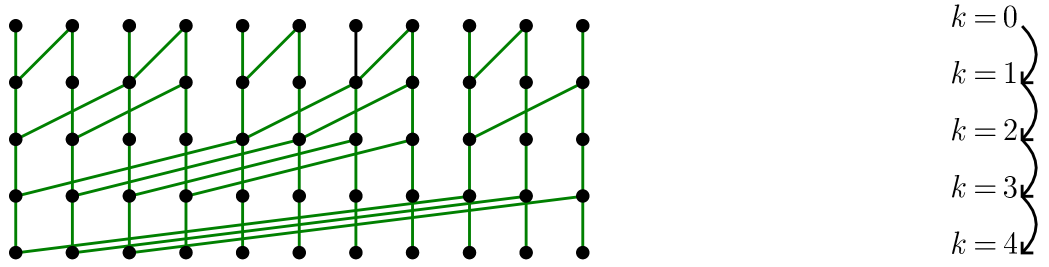

Figure 3. Schematic representation of the conversion Newton to monomial for $d=11$

The pseudo-codes for the conversion from Newton basis to monomial basis and the inverse transformation are as follows:

Algorithm TFT-Newton-to-Monomial

INPUT: an array p of length d that contains the coefficients $p_{i,1}$ w.r.t. the Newton basis
OUTPUT: the same array p now contains the coefficients $p_{i,0}$ w.r.t. the monomial basis

```
for $k=0,\ldots,q-1$
    $m=d \operatorname{div} 2^{k+1}$, $r=d \bmod 2^{k+1}$
    for $i=0,\ldots,m-1$
        for $j=0,\ldots,2^k-1$
            $p[2^{k+1}i+j] \mathrel{-}= \zeta^{[2i]_q} p[2^{k+1}i+j+2^k]$
    for $j=0,\ldots,r-2^k-1$
        $p[2^{k+1}i+j] \mathrel{-}= \zeta^{[2i]_q} p[2^{k+1}i+j+2^k]$
```

Algorithm TFT-Monomial-to-Newton

INPUT: an array p of length d that contains the coefficients $p_{i,0}$ w.r.t. the monomial basis
OUTPUT: the same array p now contains the coefficients $p_{i,1}$ w.r.t. the Newton basis

```
for $k=q-1,\ldots,0$
```

```
$m = d \operatorname{div} 2^{k+1}$, $r = d \bmod 2^{k+1}$
for $i = 0, \ldots, m-1$
  for $j = 0, \ldots, 2^k - 1$
    $p[2^{k+1} i + j] \mathrel{+}= \zeta^{[2i]_q} p[2^{k+1} i + j + 2^k]$
for $j = 0, \ldots, r - 2^k - 1$
  $p[2^{k+1} i + j] \mathrel{+}= \zeta^{[2i]_q} p[2^{k+1} i + j + 2^k]$
```

We deduce the following complexity result for the conversions, which refines Lemma 1; it is of the form $O(d \log d)$, with a tight control on the constants.

LEMMA 4. *Using the TFT evaluation points, for $\varepsilon \in \{0, 1\}$, given $P \dashv (N_{i,\varepsilon})_{i<d}$, one can compute $P \dashv (N_{i,1-\varepsilon})_{i<d}$ using $q \lfloor d/2 \rfloor$ additions or subtractions, and $q \lfloor d/2 \rfloor$ multiplications by roots of unity, with $q = \lceil \log_2 d \rceil$.*

Proof. For any given k, we do $2^k m + \max(r - 2^k, 0)$ additions/subtractions and multiplications. This is at most $\lfloor d/2 \rfloor$. □

The equivalent of Lemma 2 for the TFT points comes by using van der Hoeven's TFT algorithms for evaluation and interpolation on the monomial basis, instead of the general algorithms.

3. PROJECTIONS AND SECTIONS

Let $I \subseteq \mathbb{N}^n$ be a finite initial segment and let $(d_1, \ldots, d_n)$ be such that I is contained in $I_{d_1,\ldots,d_n}$. We present here some geometric operations on I that will be useful for the evaluation and interpolation algorithms.

Projections. We will denote by $I' \subseteq \mathbb{N}^{n-1}$ the projection

$$I' = \{\boldsymbol{i}' = (i_2, \ldots, i_n) \in \mathbb{N}^{n-1} : (0, i_2, \ldots, i_n) \in I\}$$

of I on the $(i_2, \ldots, i_n)$-coordinate plane. For $\boldsymbol{i}'$ in I', we let $d(\boldsymbol{i}') \geqslant 1$ be the unique integer such that $(d(\boldsymbol{i}') - 1, i_2, \ldots, i_n) \in I$ and $(d(\boldsymbol{i}'), i_2, \ldots, i_n) \notin I$. In particular, $d(\boldsymbol{i}') \leqslant d_1$ holds for all $\boldsymbol{i}'$.

In Figure 1, we have $d_1 = 5$; I' consists of the points of ordinates $0, 1, 2, 3$ on the vertical axis, with $d(0) = 5$, $d(1) = 4$, $d(2) = 2$ and $d(3) = 1$.

Finally, if $v = (v_1, \ldots, v_n)$ is a collection of points as defined in the introduction, with $v_k \in \mathcal{C}^{d_k}$ for all $k \leqslant n$, then we will write $v' = (v_2, \ldots, v_n)$.

Sections. For $j_1 < d_1$, we let I_{j_1} be the section

$$I_{j_1} = \{(i_1, \ldots, i_n) \in I : i_1 = j_1\}$$

and we let I'_{j_1} be the projection of I_{j_1} on the $(i_2, \ldots, i_n)$-coordinate plane. In other words, $\boldsymbol{i}' = (i_2, \ldots, i_n)$ is in I'_{j_1} if and only if $(j_1, i_2, \ldots, i_n)$ is in I. We have the following equivalent definition

$$I'_{j_1} = \{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I' : d(\boldsymbol{i}') > j_1\}.$$

Because the sets I_{j_1} form a partition of I, we deduce the equality $|I| = \sum_{j_1=0}^{d_1-1} |I'_{j_1}|$. Notice also that all I'_{j_1} are initial segments in $\mathbb{N}^{n-1}$. In Figure 1, we have $I'_0 = \{0, 1, 2, 3\}$, $I'_1 = \{0, 1, 2\}$, $I'_2 = \{0, 1\}$, $I'_3 = \{0, 1\}$ and $I'_4 = \{0\}$.

4. MULTIVARIATE BASES

From now on, we focus on multivariate polynomials. In all this section, we fix a finite initial segment $I \subseteq \mathbb{N}^n$ and $d_1, \ldots, d_n$ such that $I \subseteq I_{d_1, \ldots, d_n}$. Naturally, polynomials in $\mathcal{C}[\boldsymbol{x}]_I$ may be written in the monomial basis $(\boldsymbol{x}^{\boldsymbol{i}})_{\boldsymbol{i} \in I}$, but we may also use the multivariate Newton basis $(N_{\boldsymbol{i},v})_{\boldsymbol{i} \in I}$, defined by

$$N_{\boldsymbol{i},v}(\boldsymbol{x}) \;=\; N_{i_1,v_1}(x_1) \cdots N_{i_n,v_n}(x_n).$$

Generalizing the univariate notation, given $\boldsymbol{\varepsilon} \in \{0, 1\}^n$, we will consider a mixed monomial-Newton basis $(N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}})_{\boldsymbol{i} \in I}$ with

$$N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}}(\boldsymbol{x}) \;=\; N_{i_1,v_1,\varepsilon_1}(x_1) \cdots N_{i_n,v_n,\varepsilon_n}(x_n).$$

As in the univariate case, we will write $P \dashv (N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}})_{\boldsymbol{i} \in I}$ to indicate that P is written on the corresponding basis.

It will be useful to rely on the following decomposition. Let P be in $\mathcal{C}[\boldsymbol{x}]_I$, written on the basis $(N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}})_{\boldsymbol{i} \in I}$. Collecting coefficients, we obtain

$$P = \sum_{\boldsymbol{i} \in I} p_{\boldsymbol{i},\boldsymbol{v},\boldsymbol{\varepsilon}} \, N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}} = \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'} P_{\boldsymbol{i}',\boldsymbol{v}',\boldsymbol{\varepsilon}}(x_1) \, N_{i_2,v_2,\varepsilon_2}(x_2) \cdots N_{i_n,v_n,\varepsilon_n}(x_n), \tag{2}$$

with $\boldsymbol{i} = (i_1, \ldots, i_n)$ and

$$P_{\boldsymbol{i}',\boldsymbol{v}',\boldsymbol{\varepsilon}}(x_1) = \sum_{i_1=0}^{d(\boldsymbol{i}')-1} p_{\boldsymbol{i},\boldsymbol{v},\boldsymbol{\varepsilon}} \, N_{i_1,v_1,\varepsilon_1}(x_1). \tag{3}$$

Keep in mind that if the indices $\boldsymbol{\varepsilon}$ and ε_i are omitted, we are using the Newton basis.

LEMMA 5. *Let $\boldsymbol{\varepsilon}$ be in $\{0,1\}^n$, and let $\boldsymbol{\varepsilon}'$ be obtained by replacing $\boldsymbol{\varepsilon}_k$ by $1 - \boldsymbol{\varepsilon}_k$ in $\boldsymbol{\varepsilon}$, for some k in $\{1, \ldots, n\}$. Let P be in $\mathcal{C}[\boldsymbol{x}]_I$. Given $P \dashv (N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}})_{\boldsymbol{i} \in I}$, one can compute $P \dashv (N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}'})_{\boldsymbol{i} \in I}$ in time*

$$O\!\left(\frac{\mathsf{M}(d_k) \log d_k}{d_k} |I| \right).$$

Proof. Using a permutation of coordinates, we reduce to the case when $k = 1$. Using the above notations, it suffices to convert $P_{\boldsymbol{i}',\boldsymbol{v}',\boldsymbol{\varepsilon}}(x_1)$ from the basis $(N_{i_1,v_1,\varepsilon_1})_{i_1 < d_1}$ to the basis $(N_{i_1,v_1,1-\varepsilon_1})$ for all $\boldsymbol{i}' \in I'$. By Lemma 1, each conversion can be done in time $O(\mathsf{M}(d(\boldsymbol{i}')) \log (d(\boldsymbol{i}')))$, so the total cost is

$$O\!\left(\sum_{\boldsymbol{i}' \in I'} \mathsf{M}(d(\boldsymbol{i}')) \log (d(\boldsymbol{i}')) \right) = O\!\left(\sum_{\boldsymbol{i}' \in I'} \frac{\mathsf{M}(d(\boldsymbol{i}')) \log (d(\boldsymbol{i}'))}{d(\boldsymbol{i}')} d(\boldsymbol{i}') \right).$$

Since the function $\mathsf{M}(d)\log(d)/d$ is increasing, we get the upper bound

$$O\left(\sum_{\boldsymbol{i}'\in I'}\frac{\mathsf{M}(d_1)\log d_1}{d_1}d(\boldsymbol{i}')\right)=O\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}\sum_{\boldsymbol{i}'\in I'}d(\boldsymbol{i}')\right);$$

the conclusion follows from the equality $\sum_{\boldsymbol{i}'\in I} d(\boldsymbol{i}')=|I|$. □

Let us write $\mathbf{0}=(0,\ldots,0)$ and $\mathbf{1}=(1,\ldots,1)$, where both vectors have length n. Then, the basis $(N_{\boldsymbol{i},v,\mathbf{0}})_{\boldsymbol{i}\in I}$ is the monomial basis, whereas the basis $(N_{\boldsymbol{i},v,\mathbf{1}})_{\boldsymbol{i}\in I}$ is the Newton basis. Changing one coordinate at a time, we obtain the following corollary, which shows how to convert from the monomial basis to the Newton basis, and back.

Lemma 6. *Let $\boldsymbol{\varepsilon}$ be in $\{0,1\}^n$ and let P be in $\mathcal{C}[\boldsymbol{x}]_I$. Given $P\dashv(N_{\boldsymbol{i},v,\boldsymbol{\varepsilon}})_{\boldsymbol{i}\in I}$, one can compute $P\dashv(N_{\boldsymbol{i},v,\mathbf{1}-\boldsymbol{\varepsilon}})_{\boldsymbol{i}\in I}$ in time*

$$O\left(\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}+\cdots+\frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|\right).$$

The remarks in Section 2 about special families of points apply here as well: if the points in v have special properties (e.g., v_k is in geometric progression, or the v_k are TFT points), the cost may be reduced (both in the geometric case and in the TFT case, we may save the factors $\log d_k$).

5. Multivariate Evaluation and Interpolation

We are now in a position to state and prove our main result.

Theorem 7. *Given (I,v,P) such that $I\subseteq I_{d_1,\ldots,d_n}$, with P written on the monomial basis of $\mathcal{C}[\boldsymbol{x}]_I$, one can evaluate P at $V(I,v)$ in time*

$$O\left(\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}+\cdots+\frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|\right).$$

Conversely, given the values of P at $V(I,v)$, one can compute the representation of P on the monomial basis of $\mathcal{C}[\boldsymbol{x}]_I$ with the same cost.

Using the bound $\mathsf{M}(d)=O(d\log d\log\log d)$, and the fact that $d_i\leqslant|I|$ holds for all i, we deduce the simplified bound $O(n\,|I|\log^2|I|\log\log|I|)$ claimed in the introduction. Remark also that our result matches the cost of the algorithm of [Pan94], which applies in the special case of evaluation-interpolation at a grid.

The input P to the evaluation algorithm is given on the monomial basis of $\mathcal{C}[\boldsymbol{x}]_I$; however, internally to the algorithm, we use the Newton basis. Thus, before entering the (recursive) evaluation algorithm, we switch once and for all to the Newton basis; this does not harm complexity, in view of Lemma 6. Similarly, the interpolation algorithm uses the Newton basis, so we convert the result to the monomial basis after we have completed the interpolation.

Remark also that if the points $v_{k,0}, \ldots, v_{k,d_k-1}$ are in geometric progression for each k, then one may eliminate the factors $\log d_k$ from the complexity bound. Using TFT evaluation points allows for similar reductions.

5.1. Setup

The algorithm follows a pattern similar to Pan's multivariate evaluation and interpolation at a grid [Pan94]: e.g. for evaluation, we evaluate at the fibers above each $\boldsymbol{i}' \in I'$, and proceed recursively with polynomials obtained from the sections I_{j_1}. Using the Newton basis allows us to alleviate the issues coming from the fact that $V(I, v)$ is not a grid.

Let $P \in \mathcal{C}[\boldsymbol{x}]_I$ be written (in the Newton basis) as in the previous section:

$$\begin{aligned} P(x_1, \ldots, x_n) &= \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'} \sum_{i_1=0}^{d(\boldsymbol{i}')-1} p_{\boldsymbol{i},\boldsymbol{v}} N_{i_1,v_1}(x_1) N_{i_2,v_2}(x_2) \cdots N_{i_n,v_n}(x_n) \\ &= \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'} P_{\boldsymbol{i}',\boldsymbol{v}'}(x_1) N_{i_2,v_2}(x_2) \cdots N_{i_n,v_n}(x_n), \end{aligned}$$

where we write $\boldsymbol{i} = (i_1, \ldots, i_n)$ and

$$P_{\boldsymbol{i}',\boldsymbol{v}'}(x_1) = \sum_{i_1=0}^{d(\boldsymbol{i}')-1} p_{\boldsymbol{i},\boldsymbol{v}} N_{i_1,v_1}(x_1).$$

To $j_1 < d_1$, we associate the $(n-1)$-variate polynomial

$$P_{j_1}(x_2, \ldots, x_n) = \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'_{j_1}} P_{\boldsymbol{i}',\boldsymbol{v}'}(v_{1,j_1}) N_{i_2,v_2}(x_2) \cdots N_{i_n,v_n}(x_n).$$

The key to our algorithms is the following proposition.

Proposition 8. *For all $j = (j_1, \ldots, j_n) \in I$, the following equality holds:*

$$P(v_{1,j_1}, \ldots, v_{n,j_n}) = P_{j_1}(v_{2,j_2}, \ldots, v_{n,j_n}).$$

Proof. First, we make both quantities explicit. The left-hand side is given by

$$P(v_{1,j_1}, \ldots, v_{n,j_n}) = \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'} \sum_{i_1=0}^{d(\boldsymbol{i}')-1} p_{\boldsymbol{i},\boldsymbol{v}} N_{i_1,v_1}(v_{1,j_1}) N_{i_2,v_2}(v_{2,j_2}) \cdots N_{i_n,v_n}(v_{n,j_n}),$$

whereas the right-hand side is

$$P_{j_1}(v_{2,j_2}, \ldots, v_{n,j_n}) = \sum_{\boldsymbol{i}' = (i_2, \ldots, i_n) \in I'_{j_1}} \sum_{i_1=0}^{d(\boldsymbol{i}')-1} p_{\boldsymbol{i},\boldsymbol{v}} N_{i_1,v_1}(v_{1,j_1}) N_{i_2,v_2}(v_{2,j_2}) \cdots N_{i_n,v_n}(v_{n,j_n}),$$

where in both cases we write $\boldsymbol{i} = (i_1, \ldots, i_n)$. Thus, to conclude, it is enough to prove that, for $\boldsymbol{i}' \in I' \setminus I'_{j_1}$, we have $N_{i_1,v_1}(v_{1,j_1}) N_{i_2,v_2}(v_{2,j_2}) \cdots N_{i_n,v_n}(v_{n,j_n}) = 0$.

Indeed, recall that the assumption $\boldsymbol{i}' \in I' \setminus I'_{j_1}$ implies $d(\boldsymbol{i}') \leqslant j_1$. On the other hand, we have $j \in I$, whence the inequality $j_1 < d(j')$, where we write $j' = (j_2, \ldots, j_n)$. In particular, we deduce $d(\boldsymbol{i}') < d(j')$, which in turn implies that $\boldsymbol{i}' \nleqslant j'$. Thus, there exists $k \in \{2, \ldots, n\}$ such that $i_k > j_k$. This implies that $N_{i_k,v_k}(v_{k,j_k}) = 0$, as requested. □

5.2. Evaluation

Given (I, v, P), with $P \in \mathcal{C}[\boldsymbol{x}]_I$ written in the Newton basis $(N_{\boldsymbol{i},v})_{\boldsymbol{i}\in I}$, we show here how evaluate P at $V(I, v)$. The algorithm is the following.

- If $n = 0$, P is a constant; we return it unchanged.
- Otherwise, we compute all values $P_{\boldsymbol{i}',\boldsymbol{v}'}(v_{1,j_1})$, for $\boldsymbol{i}' \in I'$ and $0 \leqslant j_1 < d(\boldsymbol{i}')$, by applying the fast univariate evaluation algorithm to each $P_{\boldsymbol{i}',\boldsymbol{v}'}$. For $0 \leqslant j_1 < d_1$, and for $\boldsymbol{i}' \in I'_{j_1}$, we have (by definition) $j_1 < d(\boldsymbol{i}')$, so we have all the information we need to form the polynomial $P_{j_1} \dashv (N_{\boldsymbol{i}',v'})_{\boldsymbol{i}' \in I'}$. Then, we evaluate recursively each P_{j_1} at $V(I'_{j_1}, v')$, for $0 \leqslant j_1 < d_1$.

Proposition 9. *The above algorithm correctly evaluates P at $V(I, v)$ in time*

$$O\left(\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1} + \cdots + \frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|\right).$$

Proof. Correctness follows directly from Proposition 8, so we can focus on the cost analysis. Let $E(n, I, d_1, \ldots, d_n)$ denote the cost of this algorithm. The former discussion shows that $E(0, I) = 0$ and that $E(n, I, d_1, \ldots, d_n)$ is the sum of two contributions:

- the cost of computing all values $P_{\boldsymbol{i}',\boldsymbol{v}'}(v_{1,j_1})$, for $\boldsymbol{i}' \in I'$ and $j_1 < d(\boldsymbol{i}')$
- the cost of the recursive calls on (I'_{j_1}, v', P_{j_1}) for $0 \leqslant j_1 < d_1$.

Lemma 2 shows that the former admits the upper bound

$$O\left(\sum_{\boldsymbol{i}' \in I'} \mathsf{M}(d(\boldsymbol{i}'))\log(d(\boldsymbol{i}'))\right);$$

as in the proof of Lemma 5, this can be bounded by

$$O\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}|I|\right).$$

As to the recursive calls, notice that all I'_{j_1} are contained in I', which is contained in $I_{d_2,\ldots,d_n}$. Thus, for some constant K, we obtain the inequality

$$E(n, I, d_1, \ldots, d_n) \leqslant \sum_{j_1 < d_1} E(n, I'_{j_1}, d_2, \ldots, d_n) + K\,\frac{\mathsf{M}(d_1)\log d_1}{d_1}|I|. \tag{4}$$

To conclude, we prove that for all n, for all $d_1, \ldots, d_n$ and for any initial segment $I \subseteq I_{d_1,\ldots,d_n}$, we have

$$E(n, I, d_1, \ldots, d_n) \leqslant K\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1} + \cdots + \frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|. \tag{5}$$

Such an inequality clearly holds for $n = 0$. Assume by induction on n that for any $d_2, \ldots, d_n$ and any initial segment $J \subseteq I_{d_2,\ldots,d_n}$, we have

$$E(n-1, J, d_2, \ldots, d_n) \leqslant K\left(\frac{\mathsf{M}(d_2)\log d_2}{d_2} + \cdots + \frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|J|. \tag{6}$$

To prove (5), we substitute (6) in (4), to get

$$\begin{aligned} E(n, I, d_1, \ldots, d_n) &\leqslant \sum_{j_1 < d_1} K \left(\frac{\mathsf{M}(d_2) \log d_2}{d_2} + \cdots + \frac{\mathsf{M}(d_n) \log d_n}{d_n} \right) |I'_{j_1}| \\ &\quad + K \frac{\mathsf{M}(d_1) \log d_1}{d_1} |I|. \end{aligned}$$

Since $\sum_{j_1 < d_1} |I'_{j_1}| = |I|$, we are done. □

5.3. Interpolation

The interpolation algorithm is obtained by reversing step-by-step the evaluation algorithm. On input, we take (I, v, F), with $F \in \mathcal{C}^I$; the output is the unique polynomial $P \dashv (N_{\boldsymbol{i},v})_{\boldsymbol{i} \in I}$ such that $F_{\boldsymbol{i}} = P(\boldsymbol{\alpha}_{\boldsymbol{i},v})$ for all $\boldsymbol{i} \in I$.

- If $n = 0$, F consists of a single entry; we return it unchanged.
- Otherwise, we recover recursively all $P_{j_1} \dashv (N_{\boldsymbol{i}',v'})_{\boldsymbol{i}' \in I'}$, for $0 \leqslant j_1 < d_1$. This is made possible by Proposition 8, which shows that we actually know the values of each P_{j_1} at the corresponding $V(I'_{j_1}, v')$. Knowing all P_{j_1} gives us the values $P_{\boldsymbol{i}',\boldsymbol{v}'}(v_{1,j_1})$ for all $\boldsymbol{i}' \in I'$ and $0 \leqslant j_1 < d(\boldsymbol{i}')$. It suffices to interpolate each $P_{\boldsymbol{i}',\boldsymbol{v}'}$ on the Newton basis $(N_{i_1,v_1})_{i_1 < d(\boldsymbol{i}')}$ to conclude.

Correctness of this algorithm is clear and the following complexity bound is proved in a similar way as in the case of evaluation.

PROPOSITION 10. *The above algorithm correctly computes $P \dashv (N_{\boldsymbol{i},v})_{\boldsymbol{i} \in I}$ in time*

$$O\left(\left(\frac{\mathsf{M}(d_1) \log d_1}{d_1} + \cdots + \frac{\mathsf{M}(d_n) \log d_n}{d_n} \right) |I| \right).$$

6. APPLICATIONS

We conclude with an application of our results to the multiplication of polynomials and power series. Let I and $d_1, \ldots, d_n$ be as above. We let $d = \max(d_1, \ldots, d_n)$, as we assume that $\mathcal{C}$ has cardinality at least d, so that we can find $v = (v_1, \ldots, v_n)$, where $v_k = (v_{k,0}, \ldots, v_{k,d_k-1})$ consists of pairwise distinct entries in $\mathcal{C}$. Let $\delta = 1 + \max\{i_1 + \cdots + i_n : (i_1, \ldots, i_n) \in I\}$, so that $d \leqslant \delta \leqslant n(d-1)$.

6.1. Multiplication of polynomials

We discuss here the case when we want to multiply two polynomials $P_1 \in \mathcal{C}[\boldsymbol{x}]_{I_1}$ and $P_2 \in \mathcal{C}[\boldsymbol{x}]_{I_2}$ with $I_1 + I_2 = I$. In this case, we may use a simple evaluation-interpolation strategy.

- Perform multi-point evaluations of P_1 and P_2 at $V(I, \boldsymbol{v})$;
- Compute the componentwise product of the evaluations;
- Interpolate the result at $V(I, \boldsymbol{v})$ to yield the product $P_1 P_2$.

By Theorem 7, this can be done in time

$$O\left(\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}+\cdots+\frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|\right)=O\left(n\,\frac{\mathsf{M}(d)\log d}{d}\,|I|\right).$$

If $\mathcal{C}$ admits at least d points in geometric progression (or at least d TFT points), then the factor $\log d$ may be removed. This result should be compared to the algorithm of [CKL89], which has complexity $O(\mathsf{M}(|I|)\log|I|)$; that algorithm applies to more general monomial supports, but under more restrictive conditions on the base field.

6.2. Multiplication of power series

Let now $\mathfrak{m}$ be the monomial ideal generated by $\{\boldsymbol{x}^{\boldsymbol{i}}: \boldsymbol{i} \notin I\}$. We discuss here the complexity of multiplication modulo in $\mathcal{C}[\boldsymbol{x}]/\mathfrak{m}$. To our knowledge, no general algorithm with a complexity quasi-linear in $|I|$ is known.

Let us first recall an algorithm of [Sch05] and show how our results enable us to improve it. Theorem 1 of [Sch05] gives an algorithm for multiplication in $\mathcal{C}[\boldsymbol{x}]/\mathfrak{m}$, that relies on the following operations:

- $O(\delta)$ multi-point evaluations at $V(I,v)$ of polynomials in $\mathcal{C}[\boldsymbol{x}]_I$;
- $|I|$ univariate power series multiplication in precision $O(\delta)$;
- $O(\delta)$ interpolations at $V(I,v)$ of polynomials in $\mathcal{C}[\boldsymbol{x}]_I$.

The paper [Sch05] does not specify how to do the evaluation and interpolation (for lack of an efficient solution); using our results, it becomes possible to fill all the gaps in this algorithm. Applying Theorem 7, without doing any simplification, we obtain a cost of

$$O\left(\left(\frac{\mathsf{M}(d_1)\log d_1}{d_1}+\cdots+\frac{\mathsf{M}(d_n)\log d_n}{d_n}\right)|I|\,\delta+|I|\mathsf{M}(\delta)\right).$$

Using the inequality $d_i \leqslant \delta$, this gives the upper bound $O(n\,\mathsf{M}(\delta)\log\delta\;|I|)$. If we can take at least d points in geometric progression in $\mathcal{C}$ (or at least d TFT points), then the upper bound reduces to $O(n\,\mathsf{M}(\delta)\,|I|)$.

6.3. Power series with total degree truncation

The most important case of truncated power series multiplication is when we truncate with respect to the total degree. In other words, we take $I=\{(i_1,\ldots,i_n): i_1+\cdots+i_n<\delta\}$. In that case, several alternative strategies to the one of the former subsection are available [LS03, vdH04, vdH05, vdHL09], and we refer to [vdH05, vdHL09] for some benchmarks.

As it turns out, one can apply the result from Section 6.1, in the special case of polynomials supported in total degree, to improve these algorithms, when $\mathcal{C}$ admits at least d points in geometric progression (or at least d TFT points). Indeed, the algorithms from [LS03, vdHL09] rely on multivariate polynomial multiplication. Using the result of Section 6.1 in these algorithms (instead of sparse polynomial multiplication), we obtain a new algorithm of time complexity $O(n\,\frac{\mathsf{M}(d)}{d}\,|I|)$ instead of $O(n\,\mathsf{M}(|I|)\log|I|)$. For constant n, this removes a factor $O(\log d)$ from the asymptotic time complexity.

To finish, we would like to point out that the present paper almost repairs an error in [vdH04, Section 5], which was first announced in [vdH05]. Indeed, it was implicitly, but mistakenly, assumed that Proposition 8 also holds for monomial bases. The present "fix" simply consists of converting to the Newton basis before evaluating, and similarly for the inverse. The asymptotic time complexity analysis from [vdH04, Section 5] actually remains valid up to a small but non trivial constant factor. When using TFT transforms in combination with the algorithms from section 2.2, we expect the constant factor to be comprised between one and three in practice.

Bibliography

[BM74] A. Borodin and R. T. Moenck. Fast modular transforms. *Journal of Computer and System Sciences*, 8:366–386, 1974.

[BP94] D. Bini and V. Y. Pan. *Polynomial and matrix computations. Vol. 1*. Birkhäuser Boston Inc., Boston, MA, 1994. Fundamental algorithms.

[BS05] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, 2005.

[CK91] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991.

[CKL89] J. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In G. Gonnet, editor, *ISSAC '89*, pages 121–128, Portland, Oregon, 1989. ACM.

[GG03] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2-nd edition, 2003.

[LS03] G. Lecerf and É. Schost. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, 5(1):1–10, September 2003.

[Mor03] F. Mora. De nugis Groebnerialium 2: Applying Macaulay's trick in order to easily write a Gröbner basis. *Applicable Algebra in Engineering, Communication and Computing*, 13(6):437–446, 2003.

[Müh88] G. Mühlbach. On multivariate interpolation by generalized polynomials on subsets of grids. *Computing*, 40(3):201–215, 1988.

[Pan94] V. Pan. Simple multivariate polynomial multiplication. *Journal of Symbolic Computation*, 18(3):183–186, 1994.

[Sau04] T. Sauer. Lagrange interpolation on subgrids of tensor product grids. *Mathematics of Computation*, 73(245):181–190, 2004.

[Sch05] É. Schost. Multivariate power series multiplication. In M. Kauers, editor, *ISSAC '05*, pages 293–300, New York, NY, USA, 2005. ACM.

[vdH02] J. van der Hoeven. Relax, but don't be too lazy. *Journal of Symbolic Computation*, 34:479–542, 2002.

[vdH04] J. van der Hoeven. The truncated Fourier transform and applications. In J. Gutierrez, editor, *ISSAC '04*, pages 290–296, Univ. of Cantabria, Santander, Spain, July 4–7 2004. ACM.

[vdH05] J. van der Hoeven. Notes on the Truncated Fourier Transform. Technical Report 2005-5, Université Paris-Sud, Orsay, France, 2005.

[vdHL09] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial multiplication. Technical Report `http://arxiv.org/abs/0901.4323v1`, Arxiv, 2009.

[Wer80] H. Werner. Remarks on Newton type multivariate interpolation for subsets of grids. *Computing*, 25(2):181–191, 1980.