On the complexity of multivariate blockwise polynomial multiplication*

by Joris van der Hoeven and Grégoire Lecerf
Laboratoire d'Informatique, UMR 7161 CNRS
École polytechnique
91128 Palaiseau Cedex, France
{vdhoeven, lecerf}@lix.polytechnique.fr

Preliminary version of January 16, 2012

ABSTRACT

In this article, we study the problem of multiplying two multivariate polynomials which are somewhat but not too sparse, typically like polynomials with convex supports. We design and analyze an algorithm which is based on blockwise decomposition of the input polynomials, and which performs the actual multiplication in an FFT model or some other more general so called "evaluated model". If the input polynomials have total degrees at most d, then, under mild assumptions on the coefficient ring, we show that their product can be computed with $\mathcal{O}(s^{1.5337})$ ring operations, where s denotes the number of all the monomials of total degree at most $2\,d$.

Keywords

sparse polynomial multiplication, multivariate power series, evaluation-interpolation, algorithm

A.M.S. subject classification

68W30, 12-04, 30B10, 42-04

1. INTRODUCTION

Let $\mathbb{A}$ be an *effective ring*, which means that we have a data structure for representing the elements of $\mathbb{A}$ and algorithms for performing the ring operations in $\mathbb{A}$, including the equality test. The complexity for multiplying two univariate polynomials with coefficients in $\mathbb{A}$ and of degree d is rather well understood [11, Chapter 8]. Let us recall that for small d, one uses the naive multiplication, as learned at high school, of complexity $\mathcal{O}(d^2)$. For moderate d, one uses the Karatsuba multiplication [21], of complexity $\mathcal{O}(d^{\log 3/\log 2})$, or some higher order Toom-Cook scheme [4, 29], of complexity $\mathcal{O}(d^\alpha)$ with $1<\alpha<\log 3/\log 2$. For large d, one uses FFT [3, 5, 27] and TFT [14, 15] multiplications, both of complexity $\mathcal{O}(d \log d)$ whenever $\mathbb{A}$ has sufficiently many 2^k-th roots of unity, or the Cantor-Kaltofen multiplication [3, 27] with complexity in $\mathcal{O}(d \log d \log\log d)$ otherwise.

Unfortunately most of the techniques known in the univariate case do not straightforwardly extend to several variables. Even for the known extensions, the efficiency thresholds are different. For instance, the naive product is generally softly optimal when *sparse representations* are used, because the size of the output can grow with the product of the sizes of the input. In fact, the nature of the input supports plays a major role in the design of the algorithms and the determination of their mutual thresholds. In this article, we focus on and analyze an intermediate approach to several variables, which roughly corresponds to an extension of the Karatsuba and Toom-Cook strategies, and which turns out to be more efficient than the naive multiplication for instance if the supports of the input polynomials are rather dense in their respective convex hulls.

1.1 Our contributions

The main idea in our blockwise polynomial multiplication is to cut the supports in blocks of size $b_1 \times \cdots \times b_n$ and to rewrite all polynomials as "block polynomials" in $z_1^{b_1}, \ldots, z_n^{b_n}$ with "block coefficients" in

$$\mathbb{B}_b := \{P \in \mathbb{A}[z_1, \ldots, z_n] : \deg_{z_1} P < b_1, \ldots, \deg_{z_n} P < b_n\}.$$

These block coefficients are then manipulated in a suitable "evaluated model", in which the multiplication is intended to be faster than with the direct naive approach. The blockwise polynomials are themselves multiplied in a naive manner. The precise algorithm is described in Section 3.1, where we also discuss the expected speedup compared to the naive product. A precise complexity analysis is subtle because it depends too much on the nature of the supports. Although the worse case bound turns out to be no better than with the naive approach, we detail, in Section 3.2, a way for quickly determining a good block size, that suits supports that are not too small, not too thin, and rather dense in their convex hull.

Our Section 4 is devoted to implementation issues. We first design a cache-friendly version of our blockwise product. Then we adapt a blockwise product for multivariate power series truncated in total degree. Finally, we discuss a technique to slightly improve the blockwise approach for small and thin supports.

In Section 5 we analyze the complexity of the blockwise product for two polynomials supported by monomials of degree at most $d-1$: if $\mathbb{A}$ contains $\mathbb{Q}$ in its center, then we show that their product can be computed with $\mathcal{O}(s^{1.5337})$ operations in $\mathbb{A}$, where s denotes the number of all the monomials of degree at most $2(d-1)$. Notice that the constant hidden behind the latter $\mathcal{O}$ does not depend neither on d nor on the number of the variables. With no hypothesis on $\mathbb{A}$, the complexity bound we reach becomes in $\mathcal{O}(s^{1.5930})$, by a direct extension of the Karatsuba algorithm.

*. 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.

Finally, we prove similar complexity results for truncated power series

1.2 Related works

Algorithms for multiplying multivariate polynomials and series have been designed since the early ages of computer algebra [6, 7, 8, 20, 28]. Even those based on the naive techniques are a matter of constant improvements in terms of data structures, memory management, vectorization and parallelization [10, 18, 23, 24, 25, 26, 30].

With a single variable z, the usual way to multiply two polynomials P and Q of degree $d-1$ with the Karatsuba algorithm begins with splitting P and Q into $P_0(z)+z^h\,P_1(z)$ and $Q_0(z)+z^h\,Q_1(z)$, where $h:=\lfloor d/2\rfloor$. Then $P_0\,Q_0$, $P_1\,Q_1$, and $(P_0+P_1)\,(Q_0+Q_1)$ are computed recursively. This approach has been studied for several variables [22], but it turns out to be efficient mainly for plain block supports, as previously discussed in [8, Section 3].

For any kind of support, there exist general purpose sparse multiplication algorithms [2, 18]. They feature quasi-linear complexity in terms of the sizes of the supports of the input and of a given superset for the support of the product. Nevertheless the logarithmic overhead of the latter algorithms is important, and alternative approaches, directly based on the truncated Fourier transform, have been designed in [14, 15, 19] for supports which are initial segments of $\mathbb{N}^n$ (i.e. sets of monomials which are complementary to a monomial ideal), with the same order of efficiency as the FFT multiplication for univariate polynomials.

To the best of our knowledge, the blockwise technique was introduced, in a somewhat sketchy manner, in [13, Section 6.3.3], and then refined in [16, Section 6]. In the case of univariate polynomials, its complexity has been analyzed in [12], where important applications are also presented.

2. CONVENTIONS AND NOTATION

Throughout this article, we use the *sparse representation* for the polynomials, and the *computation tree model* with the *total complexity point* of view [1, Chapter 4] for the complexity analysis of the algorithms. Informally speaking, this means that complexity estimates charge a constant cost for each arithmetic operation and the equality test in $\mathbb{A}$, and that all the constants are thought to be freely at our disposal.

2.1 Block polynomials

Consider a multivariate polynomial $P\in\mathbb{A}[z]=\mathbb{A}[z_1,\ldots,z_n]$, also written as

$$P=\sum_{i\in\mathbb{N}^n}P_i\,z^i=\sum_{i_1,\ldots,i_n\in\mathbb{N}}P_{i_1,\ldots,i_n}\,z_1^{i_1}\cdots z_n^{i_n}.$$

We define its *support* by $\operatorname{supp}P=\{i\in\mathbb{N}^n\colon P_i\neq 0\}$, and denote its cardinality $|\operatorname{supp}P|$ by s_P. We interpret s_P as the *sparse size* of P.

Given a vector $b=(b_1,\ldots,b_n)\in(\mathbb{N}^>)^n$ of positive integers, we define the set of *block coefficients* at order b by

$$\mathbb{B}_b\;=\;\{P\in\mathbb{A}[z_1,\ldots,z_n]\colon \deg_{z_1}P<b_1,\ldots,\deg_{z_n}P<b_n\}.$$

Any polynomial P can uniquely be rewritten as a *block polynomial* $\bar P=\sum_{\bar i\in\mathbb{N}^n}\bar P_{\bar i}\,z^{b\bar i}\in\mathbb{B}_b[z^b]=\mathbb{B}_b[z_1^{b_1},\ldots,z_n^{b_n}]$, where

$$\bar P_{\bar i}\;=\sum_{0\leqslant i_1<b_1,\ldots,0\leqslant i_n<b_n}P_{b_1\bar i_1+i_1,\ldots,b_n\bar i_n+i_n}\,z^i.$$

Given $\bar P,\bar Q\in\mathbb{B}_b[z^b]$, we also notice that $\bar P\bar Q\in\mathbb{B}_{2b-1}[z^b]$. We let $s_b=b_1\cdots b_n$ represent the size of the block b.

2.2 Evaluation-interpolation schemes

A univariate evaluation-interpolation scheme in size $d\in\mathbb{N}^>$ on $\mathbb{A}$ is the data of

- an algorithm for computing an *evaluation* function $\mathrm{Eval}_d\colon\mathbb{B}_{d-1}\to\mathbb{A}^{\mathsf{N}(d)}$, where $\mathbb{B}_d$ is the subset of polynomials in $\mathbb{A}[z]$ of degree at most $d-1$, and where $\mathsf{N}(d)$ can be seen as the number of evaluation points, and
- an algorithm for computing an *interpolation* function, written Eval_d^{-1}, but which is not necessarily exactly the inverse of Eval_d, $\mathrm{Eval}_d^{-1}\colon\mathbb{A}^{\mathsf{N}(d)}\to\mathbb{B}_{2d-1}$,

such that Eval_d and Eval_d^{-1} are linear and

$$\mathrm{Eval}_d^{-1}(\mathrm{Eval}_d(P)\odot\mathrm{Eval}_d(Q))=PQ$$

holds for all P and Q in $\mathbb{B}_d$, where $\odot$ denotes the entry-wise product in $\mathbb{A}^{\mathsf{N}(d)}$. We write $\mathsf{E}(d)$ for a common bound on the complexity of Eval_d and Eval_d^{-1}, so that two polynomials in $\mathbb{B}_d$ can be multiplied in time $3\,\mathsf{E}(d)+\mathsf{N}(d)$. For convenience we still write Eval_d and Eval_d^{-1} for extensions to any $\mathbb{A}^k$ seen as a ring endowed with the entry-wise product.

Example 1. The Karatsuba algorithm for polynomials of degree 1 corresponds to $d=2$, $\mathsf{N}(2)=3$,

$$\mathrm{Eval}_2\colon P_0+P_1\,z\mapsto(P_0,P_0+P_1,P_1),$$

$$\mathrm{Eval}_2^{-1}\colon(C_0,C_1,C_2)\mapsto C_0+(C_1-C_0-C_2)\,z+C_2\,z^2.$$

Eval_2 can be interpreted as the evaluation of a polynomial at the values 0, 1, and $+\infty$. By induction, the Karatsuba algorithm for polynomials of degree at most $d-1$ corresponds to $\mathsf{N}(d)=3\,\mathsf{N}(\lceil d/2\rceil)$, $\mathrm{Eval}_d\colon P\mapsto\mathrm{Eval}_{\lceil d/2\rceil}(\bar P)$, where $\bar P$ is the block polynomial of P with respect to $b=(2)$, and where $\mathrm{Eval}_{\lceil d/2\rceil}(\bar P)$ actually corresponds to applying the extension of $\mathrm{Eval}_{\lceil d/2\rceil}$ to $\mathbb{B}_b[z^b]\to\mathbb{A}^{3\mathsf{N}(\lceil d/2\rceil)}$. This yields $\mathsf{N}(2^l)=3^l$ and $\mathsf{E}(2^l)=\mathcal{O}(3^l)$.

Univariate evaluation-interpolation schemes can be extended to several variables as follows: if $b=(b_1,\ldots,b_n)\in(\mathbb{N}^>)^n$ then we define $\mathsf{N}(b)=\mathsf{N}(b_1)\cdots\mathsf{N}(b_n)$, define the maps

$$\begin{aligned}\mathrm{Eval}_b\colon\mathbb{B}_b&\to\mathbb{A}^{\mathsf{N}(b)}\\ P&\mapsto\mathrm{Eval}_{b_1,1}\circ\mathrm{Eval}_{b_2,2}\circ\cdots\circ\mathrm{Eval}_{b_n,n}(P),\\ \mathrm{Eval}_b^{-1}\colon\mathbb{A}^{\mathsf{N}(b)}&\to\mathbb{B}_b\\ C&\mapsto\mathrm{Eval}_{b_n,1}^{-1}\circ\mathrm{Eval}_{b_{n-1},n-1}^{-1}\circ\cdots\circ\mathrm{Eval}_{b_1,1}^{-1}(C),\end{aligned}$$

and take

$$\mathsf{E}(b)=\mathsf{N}(b)\left(\frac{\mathsf{E}(b_1)}{b_1}+\cdots+\frac{\mathsf{E}(b_n)}{b_n}\right),\tag{1}$$

where $\mathrm{Eval}_{b_i,i}$ and $\mathrm{Eval}_{b_i,i}^{-1}$ represent the evaluation and interpolation with respect to the variable z_i:

$$\begin{aligned}\mathrm{Eval}_{b_i,i}\;:\;&\mathbb{A}^{\mathsf{N}(b_{i+1})\cdots\mathsf{N}(b_n)}\otimes\mathbb{B}_{(b_1,\ldots,b_i)}\\ &\to\mathbb{A}^{\mathsf{N}(b_i)\cdots\mathsf{N}(b_n)}\otimes\mathbb{B}_{(b_1,\ldots,b_{i-1})},\\ \mathrm{Eval}_{b_i,i}^{-1}\;:\;&\mathbb{A}^{\mathsf{N}(b_i)\cdots\mathsf{N}(b_n)}\otimes\mathbb{B}_{(2b_1,\ldots,2b_{i-1})}\\ &\to\mathbb{A}^{\mathsf{N}(b_{i+1})\cdots\mathsf{N}(b_n)}\otimes\mathbb{B}_{(2b_1,\ldots,2b_{i-1},2b_i)}.\end{aligned}$$

In the sequel we will only use multivariate evaluation-interpolation schemes constructed in this way.

3. THE BASIC ALGORITHM

For what follows, we first assume that we have a strategy for picking a suitable evaluation-interpolation scheme for any given block size $b \in (\mathbb{N}^>)^n$, and that we have a fast way to compute the corresponding functions $\mathsf{N}(b)$ and $\mathsf{E}(b)$, at least approximately. We also assume that $\mathsf{N}(d)/d$ and $\mathsf{E}(d)/d$ are increasing functions.

3.1 Blockwise multiplication

Let us first assume that we have fixed a block size b. The first version of our algorithm for multiplying two multivariate polynomials $P, Q \in \mathbb{A}[z]$ summarizes as follows:

Algorithm block-multiply(P, Q, b)
INPUT: $P, Q \in \mathbb{A}[z]$ and $b \in (\mathbb{N}^>)^n$.
OUTPUT: $PQ \in \mathbb{A}[z]$.

1. Rewrite P and Q as block polynomials $\bar{P}, \bar{Q} \in \mathbb{B}_b[z^b]$.
2. Compute $\hat{P} := \mathrm{Eval}_b\, \bar{P} := \sum_{\bar{\imath}} \mathrm{Eval}_b(\bar{P}_{\bar{\imath}})\, (z^b)^{\bar{\imath}}$ and $\hat{Q} := \mathrm{Eval}_b\, \bar{Q}$.
3. Multiply $\hat{R} := \hat{P}\hat{Q} \in \mathbb{A}^{\mathsf{N}(b)}[z^b]$ using the naive algorithm.
4. Compute $\bar{R} := \mathrm{Eval}_b^{-1}\big(\hat{R}\big) := \sum_{\bar{\imath}} \mathrm{Eval}_b^{-1}(\bar{R}_{\bar{\imath}})\, (z^b)^{\bar{\imath}} \in \mathbb{B}_{2b-1}[z^b]$.
5. Reinterpret $\bar{R}$ as a polynomial $R \in \mathbb{A}[z]$ and return R.

Let $s^*_{\bar{R}} = |\mathrm{supp}\, \bar{P} + \mathrm{supp}\, \bar{Q}| \geqslant s_{\bar{R}}$ represent the size of $\bar{R}$ if there is no coefficient of $\bar{R}$ which accidentally vanishes.

PROPOSITION 2. *The algorithm* block-multiply *works correctly as specified, and performs at most*

$$\mathsf{N}(b)\, s_{\bar{P}} s_{\bar{Q}} + \mathsf{E}(b)\, (s_{\bar{P}} + s_{\bar{Q}} + 2\, s^*_{\bar{R}})$$

operations in $\mathbb{A}$.

PROOF. The correctness is clear from the definitions. Step 1 performs no operation in $\mathbb{A}$. Steps 2 and 4 require $\mathsf{E}(b)\, (s_{\bar{P}} + s_{\bar{Q}} + s^*_{\bar{R}})$ operations. Step 3 requires $\mathsf{N}(b)\, s_{\bar{P}} s_{\bar{Q}}$ operations. Step 5 takes at most $2^k\, s^*_{\bar{R}} \leqslant \mathsf{E}(b)\, s^*_{\bar{R}}$ operations in order to add up overlapping block coefficients, where $k = |\{i \in \{1, ..., n\} : b_i > 1\}|$. □

Unfortunately, without any structural knowledge about the supports of $\bar{P}$ and $\bar{Q}$, the quantity $s^*_{\bar{R}}$ requires a time $s_{\bar{P}} s_{\bar{Q}}$ to be computed. In the next subsection we explain a heuristic approach to find a good block size b.

3.2 Heuristic computation of the block size

In order to complete our multiplication algorithm, we must explain how to set the block size. Computing quickly the block size that minimizes the number of operations in the product of two given polynomials P and Q looks like a difficult problem. In Section 5 we analyze it for asymptotically large simplex supports, but, in this subsection, we describe a heuristic for the general case.

For approximating a good block size we first need to approximate the complexity in Proposition 2 by a function, written $T(P, Q, b)$, which is intended to be easy to compute in terms of $\mathsf{N}(b)$, $\mathsf{E}(b)$, and the input supports. For instance one may opt for the formula

$$T(P, Q, b) \;=\; (\mathsf{N}(b) + 2\,\mathsf{E}(b))\, (s_{\bar{P}} + 1)\, (s_{\bar{Q}} + 1).$$

Nevertheless, if P and Q are expected to be not too sparse, then one may assume $s_{\bar{R}} \lesssim 2^n\, (s_{\bar{P}} + s_{\bar{Q}})$ and use the formula

$$T(P, Q, b) \;=\; \mathsf{N}(b)\, s_{\bar{P}} s_{\bar{Q}} + 2^{n+1}\, \mathsf{E}(b)\, (s_{\bar{P}} + s_{\bar{Q}}).$$

Then we begin the approximating process with $b = (1, ..., 1)$, and keep doubling the entry of b which reduces $T(P, Q, b)$ as much as possible. We stop as soon as $T(P, Q, b)$ cannot be further reduced in this way. Given $b \in (\mathbb{N}^>)^n$ and $i \in \{1, ..., n\}$, we write $2_i(b)$ for $(b_1, ..., b_{i-1}, 2\, b_i, b_{i+1}, ..., b_n)$. In order to make explicit the dependency in b during the execution of the algorithm, we write $P^{[b]}$ instead of $\bar{P}$.

Algorithm block-size(P, Q)
INPUT: $P, Q \in \mathbb{A}[z]$.
OUTPUT: a block size $b \in (\mathbb{N}^>)^n$ for multiplying P and Q.

Set $b := (1, ..., 1)$.
While $P^{[b]}$ and $Q^{[b]}$ are supported by at least two monomials repeat:
  Let $i \in \{1, ..., n\}$ be such that $T(P, Q, 2_i(b))$ is minimal.
  If $T(P, Q, 2_i(b)) \geqslant T(P, Q, b)$, then return b.
  Set $b := 2_i(b)$.

In the computation tree model over $\mathbb{A}$, the block-size algorithm actually performs no operation in $\mathbb{A}$. Its complexity must be analyzed in terms of *bit-operations*, which means here computation trees over $\mathbb{Z}/2\,\mathbb{Z}$. For this purpose we assume that the supports are represented by vectors of monomials, with each monomial being stored as a vector of integers in binary representation.

PROPOSITION 3. *If P and Q have total degrees at most d, then the algorithm* block-size *performs $\mathcal{O}(n^2\, (s_P + s_Q) \log^2 d)$ bit-operations, plus $\mathcal{O}(n^2 \log d)$ calls to the function T.*

PROOF. To run the algorithm block-size efficiently, we maintain the sets $S_{P,b} := \mathrm{supp}\, P^{[b]}$ and $S_{Q,b} := \mathrm{supp}\, Q^{[b]}$ throughout the main loop. Since $S_{P,2^i(b)}$ can be computed from $S_{P,b}$ with $\mathcal{O}(s_{P^{[b]}} \log d)$ bit-operations, each iteration requires at most $\mathcal{O}(n\, (s_{P^{[b]}} + s_{Q^{[b]}}) \log d)$ bit-operations. The number of iterations is bounded by $\mathcal{O}(n \log d)$. □

Remark that in favorable cases the first few iterations are expected to approximately halve $s_{P^{[b]}}$ at each stage. The expected total complexity thus becomes closer to $\mathcal{O}(n\, (s_P + s_Q) \log d)$.

4. VARIATIONS

4.1 Implementation issues

Several problems arise if one tries to implement the basic blockwise multiplication algorithm from Section 3.1 without any modification:

- The mere storage of $\hat{P}$, $\hat{Q}$ and $\hat{R}$ involves $\big(s_{\hat{P}} + s_{\hat{Q}} + s^*_{\hat{R}}\big)\, \mathsf{N}(b)$ coefficients in $\mathbb{A}$. This number is usually much larger than $s_P + s_Q + s^*_R$.
- Coefficients in $\mathbb{A}^{\mathsf{N}(b)}$ usually will not fit into cache memory, which makes the algorithm highly cache inefficient.

Both drawbacks can be removed by using a recursive multiplication algorithm instead, where the evaluation-interpolation technique is applied sequentially with respect to $z_{i_1}, ..., z_{i_k}$ for suitable pairwise distinct $i_1, ..., i_k \in \{1, ..., n\}$.

Algorithm improved-block-multiply$(P, Q, b, i_1, ..., i_k)$
INPUT: $P, Q \in \mathbb{A}[z]$, $b \in (\mathbb{N}^>)^n$ and pairwise distinct $i_1, ..., i_k \in \{1, ..., n\}$.

OUTPUT: $PQ \in \mathbb{A}[z]$.

1. If $k = 0$ then return block-multiply(P, Q, b).
2. Let $c := \big(1, \overset{(i_1-1)\times}{\ldots}, 1, b_{i_1}, 1, \ldots, 1\big)$ and rewrite P and Q as block polynomials $\bar{P}, \bar{Q} \in \mathbb{B}_c[z^c]$.
3. Compute $\hat{P} := \mathrm{Eval}_c\, \bar{P}$ and $\hat{Q} := \mathrm{Eval}_c\, \bar{Q}$.
4. For each $j \in \{1, \ldots, \mathsf{N}(c)\}$ do
   $\hat{R}_j :=$ improved-block-multiply$\big(\hat{P}_j,\ \hat{Q}_j,\ b,\ i_2,\ \ldots,\ i_k\big)$, where $\hat{R}_j$ is identified to $\sum_{\bar{\imath} \in \mathbb{N}^n} (\hat{R}_{\bar{\imath}})_j\, (z^c)^{\bar{\imath}}$.
5. Compute $\bar{R} := \mathrm{Eval}_c^{-1}\big(\hat{R}\big) \in \mathbb{B}_{2c-1}[z^c]$.
6. Reinterpret $\bar{R}$ as a polynomial $R \in \mathbb{A}[z]$ and return R.

It is not hard to check that the improved algorithm has the same complexity as the basic algorithm in terms of the number of operations in $\mathbb{A}$. Let us now examine how to choose $i_1, \ldots, i_k$ in order to increase the performance. Setting $c = (c_1, \ldots, c_n)$ with

$$c_j \ = \ \begin{cases} 1 & \text{if } j \in \{i_1, \ldots, i_k\} \\ b_j & \text{otherwise} \end{cases},$$

we first choose the set $\{i_1, \ldots, i_k\}$ in such a way that $\vartheta\, \mathsf{N}(c)$ constants in $\mathbb{A}$ fit into cache memory for a certain threshold $\vartheta \geqslant 1$. The idea is that the same coefficients should be reused as much as possible in the inner multiplication, so ϑ should not be taken to small, e.g. $\vartheta \geqslant 64$. For a fixed set $\{i_1, \ldots, i_k\}$ of indexes, we next take $i_1, \ldots, i_k$ such that $b_{i_1} \geqslant \cdots \geqslant b_{i_k}$, thereby minimizing the memory consumption of the algorithm.

4.2 Truncated multiplication

Let $P \in \mathbb{A}[z]$ and $S \subseteq \mathbb{N}^n$. We define the *truncation* of P_S to S by $P_S := \sum_{i \in S} P_i\, z^i$. Notice that $P_S = P$ if, and only if, $P \in \mathbb{A}[z]_S$ with $\mathbb{A}[z]_S := \{P \in \mathbb{A}[z] \colon \mathrm{supp}\, P \subseteq S\}$. It is sometimes convenient to represent finite sets $S \subseteq \mathbb{N}^n$ by polynomials $U \in \mathbb{A}[z]$ with $\mathrm{supp}\, U = S$, for instance by taking $U = \mathbb{u}_S := \sum_{i \in S} u\, z^i$, for a given $u \in \mathbb{A} \setminus \{0\}$. Given $P, Q \in \mathbb{A}[z]$ and a fixed support $S \subseteq \mathbb{N}^n$, the computation of $(P\,Q)_S$ is called the *truncated multiplication*. The basic blockwise multiplication algorithm is adapted as follows:

Algorithm truncated-block-multiply(P, Q, U, b)
INPUT: $P, Q, U \in \mathbb{A}[z]_{\mathrm{supp}\, U}$ and $b \in (\mathbb{N}^>)^n$.
OUTPUT: $(PQ)_{\mathrm{supp}\, U} \in \mathbb{A}[z]_{\mathrm{supp}\, U}$.

1. Rewrite P, Q and U as block polynomials $\bar{P}$, $\bar{Q}$, $\bar{U} \in \mathbb{B}_b[z^b]$.
2. Compute $\hat{P} := \mathrm{Eval}_b\, \bar{P}$ and $\hat{Q} := \mathrm{Eval}_b\, \bar{Q}$.
3. Multiply $\hat{R} := \big(\hat{P}\ \hat{Q}\big)_{\mathrm{supp}\, \bar{U}} \in \mathbb{A}^{\mathsf{N}(b)}[z^b]$ using a naive algorithm.
4. Compute $\bar{R} := \mathrm{Eval}_b^{-1}\big(\hat{R}\big) \in \mathbb{B}_{2b-1}[z^b]_{\mathrm{supp}\, \bar{U}}$.
5. Reinterpret $\bar{R}$ as a polynomial $R \in \mathbb{A}[z]$ and return $R_{\mathrm{supp}\, U}$.

In a similar way as in Proposition 2, we can show that truncated-block-multiply requires at most

$$\mathsf{N}(b)\, s_{\bar{P}} s_{\bar{Q}} + \mathsf{E}(b)\, (s_{\bar{P}} + s_{\bar{Q}} + 2\, s_{\bar{U}}) \tag{2}$$

operations in $\mathbb{A}$. However, this bound is very pessimistic since $S = \mathrm{supp}\, U$ usually has a special form which allows us to perform the naive truncated multiplication in step 3 in much less than $s_{\bar{P}} s_{\bar{Q}}$ operations. For instance, in the special and important case of truncated power series at order d, we have

$$S = S_{n,d} \ := \ \{i \in \mathbb{N}^n \colon i_1 + \cdots + i_n \leqslant d-1\},$$

and, by [16, Section 6], $|S_{n,d}| = \binom{n+d-1}{n}$. The naive truncated power series multiplication at order d can be performed using only

$$\begin{aligned} M_{n,d} \ &:= \ \sum_{k=0}^{d-1} \sum_{l_1+l_2=k} |S_{n,l_1}|\, |S_{n,l_2}| = \binom{2\,n+d-1}{2\,n} = |S_{2n,d}| \\ &\ll \ |S_{n,d}|^2 = \binom{n+d-1}{n}^2 \end{aligned}$$

operations, as shown in [16, Section 6]. Hence, if $S = S_{n,d}$ and $b = (\beta, \ldots, \beta)$, so that $\bar{S} = \mathrm{supp}\, \bar{U} = S_{n,\lceil d/\beta \rceil}$, then $s_{\bar{P}}\, s_{\bar{Q}}$ can be replaced by $\binom{2\,n+\lceil d/\beta \rceil - 1}{2\,n}$ in our complexity bound (2).

Assuming that we have a way to estimate the number of operations in $\mathbb{A}^{\mathsf{N}(b)}$ necessary to compute the truncated product $\hat{R}$ using the naive algorithm, we can adapt the heuristic of Section 3.2 to determine a candidate block size. Finally let us mention that the additional implementation tricks from Section 4.1 also extend to the truncated case in a straightforward way.

4.3 Separate treatment of the border

If we increase the block size b in block-multiply, then the block coefficients $\bar{P}_{\bar{\imath}}, \bar{Q}_{\bar{\imath}}$ get sparser and sparser. At a certain point, this makes it pointless to further increase b. One final optimization which can often be applied is to decompose $P = P_{\mathrm{int}} + P_{\mathrm{border}}$ and $Q_{\mathrm{int}} + Q_{\mathrm{border}}$ in a such a way that the multiplication $P_{\mathrm{int}}\, Q_{\mathrm{int}}$ can be done using a larger block size than the remaining part $P_{\mathrm{int}}\, Q_{\mathrm{border}} + P_{\mathrm{border}}\, Q_{\mathrm{int}} + P_{\mathrm{border}}\, Q_{\mathrm{border}}$ of the multiplication $P\,Q$. Instead of providing a full and rather technical algorithm, let us illustrate this idea with the example

$$P \ := \ Q \ := \ 1 + z_1 + z_1^2 + (1 + z_1)\, z_2 + z_2^2.$$

The naive multiplication performs 36 products in $\mathbb{A}$. The direct use of the Karatsuba algorithm with $b = (2, 2)$, reduces to the naive product of two polynomials of 3 terms with coefficients in $\mathbb{A}^9$, which amounts to $3 \times 3 \times 9 = 81$ multiplications in $\mathbb{A}$. If we decompose P and Q into

$$\begin{aligned} P_{\mathrm{int}} = Q_{\mathrm{int}} \ &= \ 1 + z_1 + (1 + z_1)\, z_2 \\ P_{\mathrm{border}} = Q_{\mathrm{border}} \ &= \ z_1^2 + z_2^2, \end{aligned}$$

then $P_{\mathrm{int}}\, Q_{\mathrm{int}}$ takes 9 products in $\mathbb{A}$, and the naive multiplication for $P_{\mathrm{int}}\, Q_{\mathrm{border}} + P_{\mathrm{border}}\, Q_{\mathrm{int}} + P_{\mathrm{border}}\, Q_{\mathrm{border}}$ uses 20 products in $\mathbb{A}$. In this example, the separate treatment of the border saves 7 products in $\mathbb{A}$ out of the 36 of the naive algorithm, and is faster than the direct use of the blockwise algorithm.

5. UNIFORM COMPLEXITY ANALYSIS

In this section we focus on the specific and important problems of multiplying polynomials $P, Q \in \mathbb{A}[z]$ of total degree at most $d-1$, and truncated power series at order d. Recall from Section 4.2 that $S_{n,d} := \{i \in \mathbb{N}^n \colon i_1 + \cdots + i_n \leqslant d-1\}$. Let

$$\begin{aligned} \lambda(\alpha) \ &:= \ (1+\alpha) \log (1+\alpha) - \alpha \log \alpha, \\ \phi_\beta(\alpha) \ &:= \ \frac{\log (2\,\beta - 1) + 2\, \lambda(\alpha/\beta)}{\lambda(2\,\alpha)}. \end{aligned}$$

Let $\eta > 0$ be the first value of α such that $\phi_1(\alpha) = \phi_4(\alpha)$, and define $\zeta := \phi_1(\eta)$. Numeric computations yield $2.1454 < \eta < 2.1455$ and $1.5336 < \zeta < 1.5337$.

5.1 Polynomial product

THEOREM 4. *Let $\varepsilon > 0$, and assume that $\mathbb{A}$ admits evaluation-interpolation schemes with $\mathsf{N}(d) = 2\,d - 1$ and $\mathsf{E}(d) = \mathcal{O}(d \log^{\nu}(d+1))$, for all $d \geqslant 1$, and for some constant $\nu \geqslant 0$. Then two polynomials in $\mathbb{A}[z]$ of degree at most $d-1$ can be multiplied with* block-multiply *using at most $\mathcal{O}(|S_{n,2d-1}|^{\zeta+\varepsilon})$ operations in $\mathbb{A}$, if the sizes of the blocks $b = (\beta, \ldots, \beta)$ are chosen as follows:*

- $\beta = 1$, *if* $d/n \leqslant \eta$,
- $\beta = 4$, *if* $\eta < d/n \leqslant 8$,
- $\beta = \lceil \alpha/2 \rceil$, *if* $8 < d/n$.

PROOF. We introduce $\alpha := d/n$. By Stirling's formula, there exists a constant K_0 such that

$$\left| \log n! - n\,(\log n - 1) + \frac{\log n}{2} \right| \leqslant K_0, \text{ for all } n \geqslant 1.$$

From $\log |S_{n,d}| = \log \binom{n+d-1}{n} = \log \left(\frac{d}{n+d} \binom{n+d}{n} \right)$, we obtain the existence of a constant K_1 such that the following inequality holds for all $n \geqslant 1$ and $d \geqslant 1$:

$$\left| \log |S_{n,d}| - n\,\lambda(\alpha) - \frac{\log n + \log(1 + n/d)}{2} \right| \leqslant K_1.$$

Therefore there exists a constant K_2 such that

$$\begin{aligned} \left| \frac{\log |S_{n,d}|}{n} - \lambda(\alpha) \right| &\leqslant K_2 \frac{\log n}{n} \\ \left| \frac{\log |S_{n,2d-1}|}{n} - \lambda(2\,\alpha) \right| &\leqslant K_2 \frac{\log n}{n} \end{aligned}$$

hold for all $d \geqslant 1$ and $n \geqslant 1$. By Proposition 2, the cost of the multiplication is bounded by $T = T_1 + T_2$ with

$$\begin{aligned} T_1 &= \mathsf{N}(b)\, s_{\bar{P}} s_{\bar{Q}} = \mathsf{N}(b)\, |S_{n,\bar{d}}|^2, \\ T_2 &= \mathsf{E}(b)\, (s_{\bar{P}} + s_{\bar{Q}} + 2\, s^*_{\bar{R}}) = 2\,\mathsf{E}(b)\, (|S_{n,\bar{d}}| + |S_{n,2\bar{d}-1}|), \end{aligned}$$

where $\bar{d} := \lceil d/\beta \rceil$, since $\deg \bar{P} \leqslant \bar{d} - 1$, $\deg \bar{Q} \leqslant \bar{d} - 1$, and $\deg \bar{R} \leqslant 2\,\bar{d} - 1$. With $\bar{\alpha} := \bar{d}/n$, $\ell(\beta) := \log(2\,\beta - 1)$, and using (1), there exists a constant K_3 such that:

$$\begin{aligned} \left| \frac{\log T_1}{n} - \ell(\beta) - 2\,\lambda(\bar{\alpha}) \right| &\leqslant K_3 \frac{\log n}{n}, \\ \left| \frac{\log T_2}{n} - \ell(\beta) - \lambda(2\,\bar{\alpha}) \right| &\leqslant K_3 \frac{\log n}{n} + \nu \frac{\log\log(\beta+1)}{n}. \end{aligned}$$

Since $\lambda'(\alpha) = \log(1 + 1/\alpha)$ and since $|\bar{\alpha} - \alpha/\beta| \leqslant 1/n$, we can bound $|\lambda(\bar{\alpha}) - \lambda(\alpha/\beta)|$ by $\frac{\log(1+n)}{n} \leqslant 2\,\frac{\log n}{n}$, and deduce the existence of an other constant K_4 such that

$$\begin{aligned} \left| \frac{\log T_1}{n} - \ell(\beta) - 2\,\lambda(\alpha/\beta) \right| &\leqslant K_4 \frac{\log n}{n}, \\ \left| \frac{\log T_2}{n} - \ell(\beta) - \lambda(2\,\alpha/\beta) \right| &\leqslant K_4 \frac{\log n}{n} + \nu \frac{\log\log(\beta+1)}{n}. \end{aligned}$$

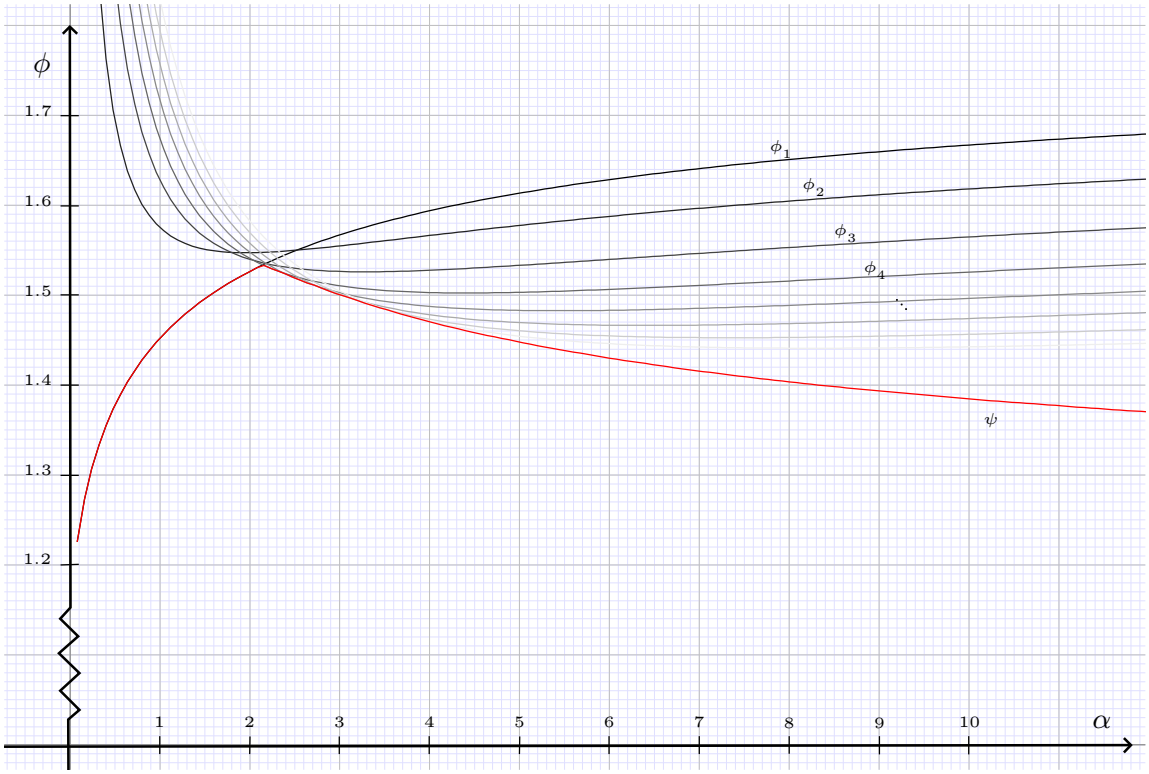

Figure 1. Illustration of the curves $\phi_\beta(\alpha)$ for various β, together with $\psi(\alpha)$.

From $\lambda'(\alpha) = \log\,(1 + 1/\alpha)$ we have that $(2\,\lambda(\alpha) - \lambda(2\,\alpha))' > 0$, and since $\lim_{\alpha\to 0}\,\lambda(\alpha) = 0$ it follows that $2\,\lambda(\alpha) - \lambda(2\,\alpha) > 0$, and that there exists a constant K_5 such that

$$\left|\frac{\log T}{n} - \ell(\beta) - 2\,\lambda(\gamma)\right| \leqslant K_5\,\frac{\log n + \log\log\,(d+1)}{n}. \qquad (3)$$

In order to express the execution time in terms of the output size, we thus have to study the function

$$\psi(\alpha) \;:=\; \min_{\beta\in\{1,2,3,\ldots\}} \phi_\beta(\alpha).$$

The first ϕ_β are plotted in Figure 1. The right limit of ϕ_1 when α tends to 0 is 1, while the same limit for the other ϕ_β is $+\infty$. Since $\lambda(\alpha) > 0$ whenever $\alpha > 0$, the sign of $\phi_\beta - \phi_1$ is the same as the one of

$$\delta_\beta(\alpha) := \ell(\beta) + 2\,\lambda(\alpha/\beta) - 2\,\lambda(\alpha).$$

From $\delta'_\beta(\alpha) = 2/\beta\,\log\,(1 + \beta/\alpha) - 2\,\log\,(1 + 1/\alpha) < 0$ and $\lim_{\alpha\to+\infty}\,\delta_\beta(\alpha) = \log\,(2\;\beta\;-\;1)\;-\;2\,\log\;\beta < 0$, it follows that there exists a unique zero, written ρ_β, of $\phi_\beta - \phi_1$ for each $\beta \geqslant 2$. These zeros can be approximated by classical ball arithmetic techniques. We used the numerix package (based on MPFR [9]) of MATHEMAGIX [17] to obtain $\rho_2 > 2.5$, $\rho_3 > 2.16$, and $\rho_4 = \eta \in (2.1454,\, 2.1455)$. Still using ball arithmetic, one checks that $\min\,(\phi_1(\alpha),\, \phi_4(\alpha)) \geqslant \psi(\alpha)$ achieves its maximum for $\alpha \in (0, 8]$ at $\alpha = \eta$. Given $\alpha > 8$, Lemma 9 of the appendix shows that $\phi_{\lceil\alpha/2\rceil}(\alpha) < \zeta$.

Finally, we have obtained so far that taking $\beta = 1$ for $\alpha \leqslant \eta$, $\beta = 4$ for $\eta < \alpha \leqslant 8$ and $\beta = \lceil\alpha/2\rceil$ for larger α, the inequality $\phi_\beta(\alpha) \leqslant \zeta$ holds for all $\alpha > 0$. Therefore there exists a constant A, such that $\log T/\log|S_{n,2d-1}| < \zeta + \varepsilon$ for all $d \geqslant 3$ and $n \geqslant A\log\log d$, or $d \leqslant 2$ and $n \geqslant A$.

It remains to bound $\log T/\log|S_{n,2d-1}|$ for $n \leqslant A\log\log d$ and d sufficiently large. There exists a constant K_6 such that:

$$\begin{aligned} |\log|S_{n,d}| - n\log d| \;&=\; \left|\sum_{i=0}^{n-1}\log\,(1 + i/d) - \log n!\right| \\ &\leqslant\; 2\,n\log n \leqslant K_6\,(\log\log d)^2, \\ |\log|S_{n,2d-1}| - n\log d| \;&\leqslant\; K_6\,(\log\log d)^2. \end{aligned}$$

Therefore, taking $\beta := \lceil\alpha/2\rceil$, there exists a contant K_7 with

$$\begin{aligned} \log T \;&\leqslant\; n\,\ell(\beta) + 2\,n\log\,(d/\beta) + \nu\log\log d + K_7\,(\log\log d)^2 \\ &\leqslant\; n\;\log\;(d/n\;+\;1)\;+\;2\;n\;\log\;(2\;n)\;+\;\nu\;\log\;\log\;d\;+ \\ &\quad\; K_7(\log\log d)^2, \end{aligned}$$

which implies that $\log T/\log|S_{n,2d-1}| \leqslant 1 + \varepsilon < \zeta$ when d is sufficiently large. □

Remark 5. In the proof of Theorem 4, it was convenient for computational purposes to explicitly chose β in terms of α. However our choice of β is suboptimal. For instance, taking $\beta = \lceil 2.5\,\alpha\rceil$ for $\alpha \geqslant 4$ is generally better whenever n remains in $\mathcal{O}(\log\log\,d)$. In fact, we think that algorithm block-size will do the choice of the block size just as well and maybe even a little bit better from the latter bounds when using a function T precise enough. A detailed proof of this claim sounds quite technical, so we have not pursued this direction any further in the present article. Nevertheless, the claim is plausible for the following reasons:

- For symmetry reasons, the block size b computed by block-size should usually be of the form $b = (2^{k+1},\, \overset{i\times}{\ldots},\, 2^{k+1},\, 2^k,\, \ldots,\, 2^k)$ for some $k \in \mathbb{N}$ (and modulo permutation of the indeterminates).
- The complexity of block-multiply for b of the above form is close to what we would obtain by taking $\beta = \sqrt[n]{b_1\cdots b_n} = 2^{k+i/n}$ in the proof of Theorem 4. When allowing for real β, the maximum of $\psi = \max_{\beta\in[1,\infty)}\,\phi_\beta(\alpha)$ is now reached at $\eta \approx 2.1438$, with $\phi_1(\eta) = \phi_\beta(\eta) = \zeta \approx 1.5336$ and $\beta \approx 3.7866$.

For rings which do not support large evaluation-interpolation schemes we propose the following extension of Theorem 4.

THEOREM 6. *Let $\varepsilon > 0$, and assume that $\mathbb{A}$ admits a unit. Then two polynomials in $\mathbb{A}[z]$ of degree at most $d-1$ can be multiplied using at most $\mathcal{O}\,(|S_{n,2d-1}|^{\zeta+\varepsilon})$ operations in $\mathbb{A}$.*

PROOF. We use the Chinese remaindering technique from [3]. It suffices to prove the theorem for $n \geqslant 2$. The unit of $\mathbb{A}$ is written 1, and we introduce $\mathbb{B}_k := \mathbb{A}[\omega]/\big(\omega^{2^k} - 1\big)$ and $\mathbb{T}_l := \mathbb{A}[\omega]/\big(\omega^{3^l} - 1\big)$. By using the aforementioned TFT algorithm, we can multiply P and Q in $\mathbb{B}_k$ (resp. in $\mathbb{T}_l$) via Theorem 4, with 2^k (resp. with 3^l) being the next power of 2 (resp. of 3) of β if we do not perform the divisions by 2 (resp. by 3). We thus obtain $2^k\,R$ and $3^l\,R$. If $u\,2^k + v\,3^l = 1$ then we deduce R as $u\,2^k\,R + v\,3^l\,R$.

Replacing all arithmetic operations in $\mathbb{A}$ by arithmetic operations in $\mathbb{B}_k \times \mathbb{T}_l$ gives rise to an additional overhead of $\mathcal{O}(\beta\log\beta\log\log\beta)$ with respect to the complexity analysis from the proof of Theorem 4. More precisely, we have a new constant K_5 such that

$$\left|\frac{\log T}{n} - \ell(\beta) - 2\,\lambda(\gamma)\right| \leqslant K_5\,\frac{\log n + \log d}{n}$$

The result is thus proved for when $n > A\log d$ for a sufficiently large constant A. It remains to examine the case when $n \leqslant A\log d$. Again, we use a similar proof as at the end of Theorem 4, with new constants K_7 and K_8 such that

$$\begin{aligned} \log T \;&\leqslant\; n\,\ell(\beta) + 2\,n\log\,(d/\beta) + \log d + K_7\,(\log\log d)^2 \\ &\leqslant\; n\;\log\;(d/n\;+\;1)\;+\;2\;n\;\log\;(2\;n)\;+\;\log\;d\;+ \\ &\quad\; K_7\,(\log\log d)^2, \\ &\leqslant\; (n+1)\log d + K_8\,n\log\log d + K_7\,(\log\log d)^2. \end{aligned}$$

Hence $\log T/\log\,|S_{n,2d-1}| \leqslant 3/2 + \varepsilon < \zeta$ for d sufficiently large. □

For rings which do not have a unit, we can use a Karatsuba evaluation-interpolation scheme. For this purpose we introduce

$$\begin{aligned} \Phi_\beta(\alpha) \;&:=\; \frac{\frac{\log 3}{\log 2}\log\beta + 2\,\lambda(\alpha/\beta)}{\lambda(2\,\alpha)}, \\ \Delta(\alpha) \;&:=\; \frac{\log 3}{2} + \lambda(\tau) - \lambda(2\,\tau). \end{aligned}$$

Let τ be the unique positive zero of Δ, and let $\zeta_2 := \Phi_{32}(32\ \tau)$. Numeric computations lead to $1.2621 < \tau < 1.2622$ and $1.5929 < \zeta_2 < 1.5930$.

THEOREM 7. *Let $\varepsilon > 0$, let $\mathbb{A}$ be any ring. Then two polynomials in $\mathbb{A}[z]$ of degree at most $d-1$ can be multiplied using at most $\mathcal{O}\left(|S_{n,2d-1}|^{\zeta_2+\varepsilon}\right)$ operations in $\mathbb{A}$.*

PROOF. We use the Karatsuba scheme [16, Section 2] with $\mathsf{N}(2^l) = 3^l$ and $\mathsf{E}(2^l) = \mathcal{O}(3^l)$ for all $l \geqslant 1$. We follow the proof of Theorem 4 and begin with a new constant K_5 such that

$$\left|\frac{\log T}{n} - \frac{\log 3}{\log 2}\log\beta - 2\,\lambda(\gamma)\right| \leqslant K_5\,\frac{\log n + \log\beta}{n}.$$

In order to express the execution time in terms of the output size, we thus have to study the function

$$\Psi(\alpha) \ := \min_{\beta\in\{1,2,4,8,\ldots\}} \Phi_\beta(\alpha).$$

The right limit of Φ_1 when α tends to 0 is 1, while the same limit for the other Φ_β is $+\infty$. Since $\lambda(\alpha) > 0$ whenever $\alpha > 0$, the sign of $\Phi_\beta - \Phi_{\beta'}$ is the same as the one of

$$\delta_{\beta,\beta'}(\alpha) := \frac{\log 3}{2\log 2}\log(\beta/\beta') + \lambda(\alpha/\beta) - \lambda(\alpha/\beta').$$

If $\beta > \beta'$, then $\delta'_{\beta,\beta'}(\alpha) < 0$ and $\lim_{\alpha\to+\infty}\delta_\beta(\alpha) = \left(\frac{\log 3}{2\log 2} - 1\right)\log(\beta/\beta') < 0$, which implies the existence of a unique zero of $\Phi_\beta - \Phi_{\beta'}$, written $\rho_{\beta,\beta'}$. In particular, with $\beta' = \beta/2$, we obtain $\rho_{\beta,\beta/2} = \beta\,\tau$. For convenience we let $\rho_\beta := \rho_{\beta,\beta/2}$, and display the first few values:

β	4	8	16	32	64	128
ρ_β	5.04855	10.0971	20.1942	40.3884	80.7768	161.554
$\Phi_\beta(\rho_\beta)$	1.57811	1.58853	1.59208	1.59297	1.59286	1.59242

One can verify that $\frac{\log 3}{2\log 2}\log(\mathrm{B}/\beta) + \lambda(\beta\,\tau/\mathrm{B}) - \lambda(\tau) = \frac{\log 3}{2\log 2}\log(1/t) + \lambda(\tau\,t) - \lambda(\tau) > 0$ holds for all $t := \beta/B < 1$, which implies that $\Phi_{\mathrm{B}}(\rho_\beta) > \Phi_\beta(\rho_\beta)$ for all $\mathrm{B} \geqslant 2\ \beta$. Therefore $\Psi(\alpha) = \Phi_\beta(\alpha)$ with β such that $\rho_\beta < \alpha \leqslant \rho_{2\beta}$ (with the convention that $\rho_0 = 0$). Lemma 10 of the appendix provides us with $\Psi(\alpha) \leqslant \Phi_{32}(\rho_{32}) = \zeta_2$. Therefore there exists a constant A, such that $\log T/\log|S_{n,2d-1}| < \zeta_2 + \varepsilon$ for all $d \geqslant 2$ and $n \geqslant A\log d$, or $d \leqslant 1$ and $n \geqslant A$.

It remains to bound $\log T/\log|S_{n,2d-1}|$ for $n \leqslant A\log d$ and d sufficiently large, as in the end of the proof of Theorem 4. In this range we take $\beta = 2^l$ with $l := \left\lfloor \frac{\log\alpha}{0.3\log 2}\right\rfloor$, so that $\beta \leqslant \alpha^{1/0.3}$. There exist constants K_7 and K_8 such that:

$$\log T \ \leqslant \ \left((n+1)\frac{\log 3}{\log 2} - 1\right)\log\beta + 2\,n\log(d/\beta) + K_7\,n\,(\log\log d)^2.$$

Therefore, using $n \geqslant 2$, we have

$$\frac{\log T}{\log|S_{n,2d-1}|} < 1.5915 < \zeta_2$$

whenever d is sufficiently large. □

5.2 Power series product

THEOREM 8. *Let $\varepsilon > 0$, and assume that $\mathbb{A}$ admits evaluation-interpolation schemes with $\mathsf{N}(d) = 2\,d-1$ and $\mathsf{E}(d) = \mathcal{O}(d\log^\nu(d+1))$, for all $d \geqslant 1$, and for some constant $\nu \geqslant 0$. Then two power series in $\mathbb{A}[z]$ truncated at order d can be multiplied with* truncated-block-multiply *using at most $\mathcal{O}\left(|S_{n,d}|^{\zeta+\varepsilon}\right)$ operations in $\mathbb{A}$, if the sizes of the blocks $b = (\beta,\ldots,\beta)$ are chosen as follows:*

- $\beta = 1$, *if* $d/n \leqslant 2\,\eta$,
- $\beta = 4$, *if* $2\,\eta < d/n \leqslant 16$,
- $\beta = \lceil d/n\rceil$, *if* $16 < d/n$.

PROOF. The proof is similar to the one of Theorem 4. In the present case, for a suitable new constant K_2 the following inequalities hold:

$$\left|\frac{\log|S_{n,d}|}{n} - \lambda(\alpha)\right| \ \leqslant \ K_2\,\frac{\log n}{n},$$
$$\left|\frac{\log|S_{2n,d}|}{n} - 2\,\lambda(\alpha/2)\right| \ \leqslant \ K_2\,\frac{\log n}{n}.$$

The cost of the multiplication is bounded by $T = T_1 + T_2$ with $T_1 = \mathsf{N}(b)\,S_{2n,\bar d}$ and $T_2 = 4\,\mathsf{E}(b)\,S_{n,\bar d}$. There exists a new constant K_4 such that:

$$\left|\frac{\log T_1}{n} - \ell(\beta) - 2\,\lambda\left(\frac{\alpha}{2\,\beta}\right)\right| \ \leqslant \ K_4\,\frac{\log n}{n},$$
$$\left|\frac{\log T_2}{n} - \ell(\beta) - \lambda\left(\frac{\alpha}{\beta}\right)\right| \ \leqslant \ K_4\,\frac{\log n}{n} + \nu\,\frac{\log\log(\beta+1)}{n}.$$

The present situation is similar to the one of the proof of Theorem 4, with $\alpha/2$ instead of α. Therefore by taking $\beta = 1$ for $\alpha < 2\,\eta$, $\beta = 4$ for $2\,\eta < \alpha \leqslant 16$ and $\beta = \lceil\alpha\rceil$ for larger α, there exists a constant A, such that $\log T/\log|S_{n,d}| < \zeta + \varepsilon$ for all $d \geqslant 3$ and $n \geqslant A\log\log d$, or $d \leqslant 2$ and $n \geqslant A$.

It remains to bound $\log T/\log|S_{n,d}|$ for $n \leqslant A\log\log d$ and d sufficiently large. There exists a new constant K_6 such that:

$$|\log|S_{n,d}| - n\log d| \ \leqslant \ K_6\,(\log\log d)^2,$$
$$|\log|S_{2n,2d-1}| - 2\,n\log d| \ \leqslant \ K_6\,(\log\log d)^2.$$

Taking $\beta := \lceil\alpha\rceil$, we thus have a contant K_7 with

$$\log T \ \leqslant \ n\,\ell(\beta) + 2\,n\log(d/\beta) + \nu\log\log d + K_7\,(\log\log d)^2$$
$$\leqslant \ n\,\log\,(d/n\,+\,1)\,+\,2\,n\,\log\,(n)\,+\,\nu\,\log\,\log\,d\,+\,K_7(\log\log d)^2.$$

We conclude that $\log T/\log|S_{n,d}| \leqslant 1 + \varepsilon < \zeta$ for d sufficiently large. □

6. REFERENCES

[1] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, 1997.

[2] J. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In *Proc. ISSAC 1989*, pages 121–128, Portland, Oregon, A.C.M., New York, 1989. ACM Press.

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

[4] S. A. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966.

[5] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Computat.*, 19:297–301, 1965.

[6] S. Czapor, K. Geddes, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[7] J. H. Davenport, Y. Siret, and É. Tournier. *Calcul formel : systèmes et algorithmes de manipulations algébriques*. Masson, Paris, France, 1987.

[8] R. Fateman. Comparing the speed of programs for sparse polynomial multiplication. *SIGSAM Bull.*, 37(1):4–15, 2003.

[9] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2), June 2007. Software available at `http://www.mpfr.org`.

[10] M. Gastineau and J. Laskar. Development of TRIP: Fast sparse multivariate polynomial multiplication using burst tries. In *Proceedings of ICCS 2006*, LNCS 3992, pages 446–453. Springer, 2006.

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

[12] G. Hanrot and P. Zimmermann. A long note on Mulders' short product. *J. Symbolic Comput.*, 37(3):391–401, 2004.

[13] J. van der Hoeven. Relax, but don't be too lazy. *J. Symbolic Comput.*, 34(6):479–542, 2002.

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

[15] J. van der Hoeven. Notes on the truncated Fourier transform. Technical Report 2005-5, Université Paris-Sud, Orsay, France, 2005.

[16] J. van der Hoeven. Newton's method and FFT trading. *J. Symbolic Comput.*, 45(8), 2010.

[17] J. van der Hoeven et al. Mathemagix. Available from `http://www.mathemagix.org`, 2002.

[18] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial multiplication. Technical Report `http://hal.archives-ouvertes.fr/hal-00476223`, École polytechnique and CNRS, 2010.

[19] J. van der Hoeven and É. Schost. Multi-point evaluation in higher dimensions. Technical report, HAL, 2010. `http://hal.archives-ouvertes.fr/hal-00477658`.

[20] S. C. Johnson. Sparse polynomial arithmetic. *SIGSAM Bull.*, 8(3):63–71, 1974.

[21] A. Karatsuba and J. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595–596, 1963.

[22] G. I. Malaschonok and E. S. Satina. Fast multiplication and sparse structures. *Programming and Computer Software*, 30(2):105–109, 2004.

[23] M. Monagan and R. Pearce. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. In *Proc. of CASC 2007*, pages 295–315. Springer, 2007.

[24] M. Monagan and R. Pearce. Parallel sparse polynomial multiplication using heaps. In *Proc. ISSAC 2009*, pages 263–270, New York, NY, USA, 2009. ACM.

[25] M. Monagan and R. Pearce. Polynomial multiplication and division in Maple 14. *Communications in Computer Algebra*, 44(4):205–209, 2010.

[26] M. Monagan and R. Pearce. Sparse polynomial pseudo division using a heap. *J. Symbolic Comput.*, 46(7):807–822, 2011.

[27] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[28] D. R. Stoutemyer. Which polynomial representation is best? In *Proceedings of the 1984 MACSYMA Users' Conference: Schenectady, New York, July 23–25, 1984*, pages 221–243, 1984.

[29] A. L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics*, 4(2):714–716, 1963.

[30] T. Yan. The geobucket data structure for polynomials. *J. Symbolic Comput.*, 25(3):285–293, 1998.

APPENDIX TECHNICAL LEMMAS

This appendix contains details on how one can bound the functions $\phi_{\lceil\alpha/2\rceil}(\alpha)$ and $\Psi(\alpha)$ used in Section 5. We first prove the bounds in an explicit neighbourhood of infinity and then we rely on certified ball arithmetic for the remaining compact set.

LEMMA 9. *For all $\alpha > 8$ we have $\phi_{\lceil\alpha/2\rceil}(\alpha) < \zeta$.*

PROOF. Let $\varphi(\alpha) := \phi_{\lceil\alpha/2\rceil}(\alpha)$ and $\tilde{\varphi}(\alpha) := \phi_{\alpha/2}(\alpha)$. We shall first show that $\tilde{\varphi}'(\alpha) < 0$ for all $\alpha > 0$. Let $\hat{\varphi}(\alpha) := (\alpha-1)\,\tilde{\varphi}'(\alpha)\,\lambda^2(2\,\alpha) = \lambda(2\,\alpha) - 2\,\lambda'(2\,\alpha)\,(\alpha-1)\,(\log(\alpha-1) + 2\,\lambda(2))$. Since $\hat{\varphi}(8) < 0$, it suffices to show that $\hat{\varphi}'(\alpha) < 0$. Since $\hat{\varphi}'(\alpha) = -2\,(2\,\lambda''(2\,\alpha)\,(\alpha-1) + \lambda'(2\,\alpha))\,(\log(\alpha-1) + 2\,\lambda(2))$, we let $\check{\varphi}(\alpha) := 2\,\lambda''(2\,\alpha)\,(\alpha-1) + \lambda'(2\,\alpha)$, and have to prove that $\check{\varphi}(\alpha) \geqslant 0$. The latter inequality is correct since $\lim_{\alpha\to+\infty} \check{\varphi}(\alpha) = 0$, and $\check{\varphi}'(\alpha) = -\frac{5\,\alpha+1}{\alpha^2\,(2\,\alpha+1)^2} \leqslant 0$ for $\alpha \geqslant 8$. On the other hand, for $\alpha \geqslant 16$ we have

$$\begin{aligned} |\varphi(\alpha) - \tilde{\varphi}(\alpha)| &\leqslant \frac{\log(2\,\lceil\alpha/2\rceil - 1) - \log(\alpha-1)}{\lambda(2\,\alpha)} \\ &\quad + \frac{2\,\lambda(2) - 2\,\lambda(\alpha/\lceil\alpha/2\rceil)}{\lambda(2\,\alpha)} \\ &\leqslant \frac{\log(1+3/15) + 2\,\big(\lambda(2) - \lambda\big(2\,\frac{16}{17}\big)\big)}{\lambda(32)} \\ &< 0.07. \end{aligned}$$

Since $\tilde{\varphi}(16) < 1.39$, we have shown that $\varphi(\alpha) < \zeta$ for $\alpha \geqslant 16$. For α between 8 and 16, we appeal to numeric computations to verify that $\varphi(\alpha) < \zeta$ still holds. □

LEMMA 10. *For all $\alpha > 0$ we have $\Psi(\alpha) \leqslant \zeta_2$.*

PROOF. Let $\kappa := \log 3/\log 2$, and $\varsigma := 2\,\lambda(\tau) - \kappa\log\tau \simeq 2.7365$. Let $\tilde{\varphi}(\alpha) := \Phi_{\alpha/\tau}(\alpha)$. We shall first show that $\tilde{\varphi}'(\alpha) < 0$ for all $\alpha \geqslant 100$. Let $\hat{\varphi}(\alpha) := \alpha\,\tilde{\varphi}'(\alpha)\,\lambda^2(2\,\alpha) = \kappa\,\lambda(2\,\alpha) - 2\,\lambda'(2\,\alpha)\,\alpha\,(\kappa\log\alpha + \varsigma)$. Since $\hat{\varphi}(100) < 0$, it suffices to show that $\hat{\varphi}'(\alpha) < 0$. Since $\hat{\varphi}'(\alpha) = -2\,(2\,\alpha\,\lambda''(2\,\alpha) + \lambda'(2\,\alpha))\,(\kappa\,\log\,\alpha\,+\,\varsigma)$, we let $\check{\varphi}(\alpha) := \alpha\,\lambda''(\alpha) + \lambda'(\alpha)$, and have to prove that $\check{\varphi}(\alpha) \geqslant 0$. The latter inequality is correct since $\lim_{\alpha\to+\infty} \check{\varphi}(\alpha) = 0$, and $\check{\varphi}'(\alpha) = -\frac{\alpha}{\alpha^2\,(2\,\alpha+1)^2} \leqslant 0$.

Let $l := \left\lfloor \frac{\log(d/n) - \log\tau}{\log 2} \right\rfloor$ and $\beta_l := 2^l$. We introduce $\varphi(\alpha) := \Phi_{\beta^l}(\alpha)$. Since $\alpha/(2\,\tau) < \beta_l \leqslant \alpha/\tau$, for $\alpha \geqslant 2^{256}$, we have

$$\begin{aligned} |\varphi(\alpha) - \tilde{\varphi}(\alpha)| &\leqslant 2\,\frac{\lambda(\alpha/\beta_l) - \lambda(\tau)}{\lambda(2\,\alpha)} \\ &\leqslant \frac{\log 3}{\lambda(2^{257})} < 0.0062. \end{aligned}$$

Since $\tilde{\varphi}(2^{256}) < 1.5853$, we have shown that $\varphi(\alpha) < \zeta_2$ for $\alpha \geqslant 2^{256}$.

For $2^{16} \leqslant \alpha < 2^{256}$, we symbolically computed $\Phi''_\beta(\alpha)$ and straightforwardly lower bounded it by a function $\Gamma(\beta)$ in the domain $\beta\,\tau \leqslant \alpha \leqslant 2\beta\,\tau$. For each value of $\beta = 2^{16}, 2^{17}, \ldots, 2^{255}$, we used ball arithmetic to show that $\Gamma(\beta) > 0$. For smaller values of α, we could directly compute that $\Phi''_\beta(\alpha) > 0$ whenever $\beta\,\tau \leqslant \alpha \leqslant 2\beta\,\tau$. Finally we computed that $\Phi_\beta(\beta\,\tau) \leqslant \zeta_2$ for all $\beta = 2, 4, 8, \ldots, 256$, which concludes the proof. □