On the complexity of multivariate polynomial division*

by Joris van der Hoeven

LIX, CNRS
École polytechnique
91128 Palaiseau Cedex
France

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

April 30, 2017

Abstract

In this paper, we present a new algorithm for reducing a multivariate polynomial with respect to an autoreduced tuple of other polynomials. In a suitable sparse complexity model, it is shown that the execution time is essentially the same (up to a logarithmic factor) as the time needed to verify that the result is correct.

Keywords: sparse reduction, complexity, division, algorithm
A.M.S. subject classification: 68W30, 12Y05, 68W40, 13P10

1 Introduction

Sparse interpolation [1, 5, 2, 13] provides an interesting paradigm for efficient computations with multivariate polynomials. In particular, under suitable hypothesis, multiplication of sparse polynomials can be carried out in quasi-linear time, in terms of the expected output size. More recently, other multiplication algorithms have also been investigated, which outperform naive and sparse interpolation under special circumstances [14, 12]. An interesting question is how to exploit such asymptotically faster multiplication algorithms for the purpose of polynomial elimination. In this paper, we will focus on the reduction of a multivariate polynomial with respect to an autoreduced set of other polynomials and show that fast multiplication algorithms can indeed be exploited in this context in an asymptotically quasi-optimal way.

Consider the polynomial ring $\mathbb{K}[x] = \mathbb{K}[x_1, \ldots, x_n]$ over an effective field $\mathbb{K}$ with an effective zero test. Given a polynomial $P = \sum_{i \in \mathbb{N}^n} P_i\, x^i = \sum_{i_1, \ldots, i_n \in \mathbb{N}} P_{i_1, \ldots, i_n}\, x_1^{i_1} \cdots x_n^{i_n}$, we call $\operatorname{supp} P = \{i \in \mathbb{N}^n : P_i \neq 0\}$ the *support* of P. The naive multiplication of two sparse polynomials $P, Q \in \mathbb{K}[x]$ requires *a priori* $\mathcal{O}(|\operatorname{supp} P|\ |\operatorname{supp} Q|)$ operations in $\mathbb{K}$. This upper bound is sharp if P and Q are very sparse, but pessimistic if P and Q are dense.

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

Assuming that $\mathbb{K}$ has characteristic zero, a better algorithm was proposed in [2] (see also [1, 5] for some background). The complexity of this algorithm can be expressed in the expected size $s = |\operatorname{supp} P + \operatorname{supp} Q|$ of the *output* (when no cancellations occur). It is shown that P and Q can be multiplied using only $\mathcal{O}(\mathsf{M}(s) \log s)$ operations in $\mathbb{K}$, where $\mathsf{M}(s) = \mathcal{O}(s \log s \log\log s)$ stands for the complexity of multiplying two univariate polynomials in $\mathbb{K}[z]$ of degrees $<s$. Unfortunately, the algorithm in [2] has two drawbacks:

1. The algorithm leads to a big growth for the sizes of the coefficients, thereby compromising its bit complexity (which is often worse than the bit complexity of naive multiplication).

2. It requires $\operatorname{supp} P\,Q \subseteq \operatorname{supp} P + \operatorname{supp} Q$ to be known beforehand. More precisely, whenever a bound $\operatorname{supp} P\,Q \subseteq \operatorname{supp} P + \operatorname{supp} Q \subseteq \mathcal{S}$ is known, then we really obtain a multiplication algorithm of complexity $\mathcal{O}(\mathsf{M}(|\mathcal{S}|) \log |\mathcal{S}|)$.

In practice, the second drawback is of less importance. Indeed, especially when the coefficients in $\mathbb{K}$ can become large, then the computation of $\operatorname{supp} P + \operatorname{supp} Q$ is often cheap with respect to the multiplication PQ itself, even if we compute $\operatorname{supp} P + \operatorname{supp} Q$ in a naive way.

Recently, several algorithms were proposed for removing the drawbacks of [2]. First of all, in [13] we proposed a practical algorithm with essentially the same advantages as the original algorithm from [2], but with a good bit complexity and a variant which also works in positive characterisic. However, it still requires a bound for $\operatorname{supp} PQ$ and it only works for special kinds of fields $\mathbb{K}$ (which nevertheless cover the most important cases such as $\mathbb{K} = \mathbb{Q}$ and finite fields). Even faster algorithms were proposed in [9, 14], but these algorithms only work for special supports. Yet another algorithm was proposed in [7, 12]. This algorithm has none of the drawbacks of [2], but its complexity is suboptimal (although better than the complexity of naive multiplication).

At any rate, these recent developments make it possible to rely on fast sparse polynomial multiplication as a building block, both in theory and in practice. This makes it natural to study other operations on multivariate polynomials with this building block at our disposal. One of the most important such operations is division.

The multivariate analogue of polynomial division is the reduction of a polynomial $A \in \mathbb{K}[x]$ with respect to an autoreduced tuple $B = (B_1, \ldots, B_b) \in \mathbb{K}[x]^b$ of other polynomials. This leads to a relation

$$A \;=\; Q_1\, B_1 + \cdots + Q_b\, B_b + R, \tag{1}$$

such that none of the terms occurring in R can be further reduced with respect to B. In this paper, we are interested in the computation of R as well as $Q_1, \ldots, Q_b$. We will call this the problem of *extended reduction*, in analogy with the notion of an "extended g.c.d.".

Now in the univariate context, "relaxed power series" provide a convenient technique for the resolution of implicit equations [6, 7, 8, 10]. One major advantage of this technique is that it tends to respect most sparsity patterns which are present in the input data and in the equations. The main technical tool in this paper (see section 3) is to generalize this technique to the setting of multivariate polynomials, whose terms are ordered according to a specific admissible ordering on the monomials. This will make it possible to rewrite (1) as a so called recursive equation (see section 4.2), which can be solved in a relaxed manner. Roughly speaking, the cost of the extended reduction then reduces to the cost of the relaxed multiplications $Q_1\ B_1$, ..., $Q_b\ B_b$. Up to a logarithmic overhead, we will show (theorem 7) that this cost is the same as the cost of checking the relation (1).

In order to simplify the exposition, we will adopt a simplified sparse complexity model throughout this paper. In particular, our complexity analysis will not take into account the computation of support bounds for products or results of the extended reduction. Bit complexity issues will also be left aside in this paper. We finally stress that our results are mainly of theoretical interest since none of the proposed algorithms have currently been implemented. Nevertheless, practical gains are not to be excluded, especially in the case of small n, high degrees and dense supports.

2 Notations

Let $\mathbb{K}$ be an effective field with an effective zero test and let x_1, ..., x_n be indeterminates. We will denote

$$\begin{aligned} \mathbb{K}[x] &= \mathbb{K}[x_1, ..., x_n] \\ P_i &= P_{i_1,...,i_n} \\ x^i &= x_1^{i_1} \cdots x_n^{i_n} \\ i \preccurlyeq j &\Leftrightarrow i_1 \leqslant j_1 \wedge \cdots \wedge i_n \leqslant j_n, \end{aligned}$$

for any $i, j \in \mathbb{N}^n$ and $P \in \mathbb{K}[x]$. In particular, $i \preccurlyeq j \Leftrightarrow x^i \,|\, x^j$. For any subset $E \subseteq \mathbb{N}^n$ we will denote by $\mathrm{Fin}(E) = \{j \in \mathbb{N}^n : \exists i \in E, i \preccurlyeq j\}$ the *final segment* generated by E for the partial ordering $\preccurlyeq$.

Let $\leqslant$ be a total ordering on $\mathbb{N}^n$ which is compatible with addition. Two particular such orderings are the lexicographical ordering $\leqslant^{\mathrm{lex}}$ and the reverse lexicographical ordering $\leqslant^{\mathrm{rlex}}$:

$$\begin{aligned} i <^{\mathrm{lex}} j &\Leftrightarrow \exists k, i_1 = j_1 \wedge \cdots \wedge i_{k-1} = j_{k-1} \wedge i_k < j_k \\ i <^{\mathrm{rlex}} j &\Leftrightarrow \exists k, i_k < j_k \wedge i_{k+1} = j_{k+1} \wedge \cdots \wedge i_n = j_n. \end{aligned}$$

In general, it can be shown [16] that there exist real vectors $\lambda_1, ..., \lambda_n \in \mathbb{R}^m$ with $m \leqslant n$, such that

$$i \leqslant j \Leftrightarrow (\lambda_1 \cdot i, ..., \lambda_m \cdot i) \leqslant^{\mathrm{lex}} (\lambda_1 \cdot j, ..., \lambda_m \cdot j). \tag{2}$$

In what follows, we will assume that $\lambda_1, ..., \lambda_n \in \mathbb{N}^n$ and $\gcd((\lambda_i)_1, ..., (\lambda_i)_n) = 1$ for all i. We will also denote

$$\lambda \cdot i \;=\; (\lambda_1 \cdot i, ..., \lambda_n \cdot i).$$

For instance, the graded reverse lexicographical ordering $\leqslant^{\text{grlex}}$ is obtained by taking $\lambda_1 = (1, ..., 1)$, $\lambda_2 = (0, ..., 1)$, $\lambda_2 = (0, ..., 0, 1, 0)$, ..., $\lambda_n = (0, 1, 0, ..., 0)$.

Given $P \in \mathbb{K}[x]$, we define its *support* by

$$\operatorname{supp} P \;=\; \{i \in \mathbb{N}^n : P_i \neq 0\}.$$

If $P \neq 0$, then we also define its *leading exponent* l_P and *coefficient* c_P by

$$\begin{aligned} l_P &= \max_{\leqslant} \operatorname{supp} P \\ c_P &= P_{l_P}. \end{aligned}$$

Given a finite set E, we will denote its cardinality by $|E|$.

3 Relaxed multiplication

3.1 Relaxed power series

Let us briefly recall the technique of relaxed power series computations, which is explained in more detail in [7]. In this computational model, a univariate power series $f \in \mathbb{K}[[z]]$ is regarded as a stream of coefficients f_0, f_1, ... When performing an operation $g = \Phi(f_1, ..., f_k)$ on power series it is required that the coefficient g_n of the result is output as soon as sufficiently many coefficients of the inputs are known, so that the computation of g_n does not depend on the further coefficients. For instance, in the case of a multiplication $h = f\,g$, we require that h_n is output as soon as $f_0, ..., f_n$ and $g_0, ..., g_n$ are known. In particular, we may use the naive formula $h_n = \sum_{i=0}^{n} f_i\, g_{n-i}$ for the computation of h_n.

The additional constraint on the time when coefficients should be output admits the important advantage that the inputs may depend on the output, provided that we add a small delay. For instance, the exponential $g = \exp f$ of a power series $f \in z\,\mathbb{K}[[z]]$ may be computed in a relaxed way using the formula

$$g \;=\; \int f'\, g.$$

Indeed, when using the naive formula for products, the coefficient g_n is given by

$$g_n \;=\; \tfrac{1}{n}\,(f_1\, g_{n-1} + 2\, f_2\, g_{n-2} + \cdots + n\, f_n\, g_0),$$

and the right-hand side only depends on the previously computed coefficients g_0, ..., g_{n-1}. More generally, equations of the form $g = \Phi(g)$ which have this property are called *recursive* equations and we refer to [11] for a mechanism to transform fairly general implicit equations into recursive equations.

The main drawback of the relaxed approach is that we cannot directly use fast algorithms on polynomials for computations with power series. For instance, assuming that $\mathbb{K}$ has sufficiently many 2^p-th roots of unity and that field operations in $\mathbb{K}$ can be done in time $\mathcal{O}(1)$, two polynomials of degrees $< n$ can be multiplied in time $\mathsf{M}(n) = \mathcal{O}(n \log n)$, using FFT multiplication [3]. Given the truncations $f_{;n} = f_0 + \cdots + f_{n-1}\, z^{n-1}$ and $g_{;n} = g_0 + \cdots + g_{n-1}\, z^{n-1}$ at order n of power series $f, g \in \mathbb{K}[[z]]$, we may thus compute the truncated product $(f\,g)_{;n}$ in time $\mathsf{M}(n)$ as well. This is much faster than the naive $\mathcal{O}(n^2)$ relaxed multiplication algorithm for the computation of $(f\,g)_{;n}$. However, the formula for $(f\,g)_0$ when using FFT multiplication depends on all input coefficients $f_0, \ldots, f_{n-1}$ and $g_0, \ldots, g_{n-1}$, so the fast algorithm is not relaxed (we will say that FFT multiplication is a *zealous* algorithm). Fortunately, efficient relaxed multiplication algorithms do exist:

Theorem 1. [6, 7, 4] *Let $\mathsf{M}(n)$ be the time complexity for the multiplication of polynomials of degrees $< n$ in $\mathbb{K}[z]$. Then there exists a relaxed multiplication algorithm for series in $\mathbb{K}[[z]]$ at order n of time complexity $\mathsf{R}(n) = \mathcal{O}(\mathsf{M}(n) \log n)$.*

Remark 2. In fact, the algorithm from theorem 1 generalizes to the case when the multiplication on $\mathbb{K}$ is replaced by an arbitrary bilinear "multiplication" $\mathbb{M}_1 \times \mathbb{M}_2 \to \mathbb{M}_3$, where $\mathbb{M}_1$, $\mathbb{M}_2$ and $\mathbb{M}_3$ are effective modules over an effective ring $\mathbb{A}$. If $\mathsf{M}(n)$ denotes the time complexity for multiplying two polynomials $P \in \mathbb{M}_1[z]$ and $Q \in \mathbb{M}_2[z]$ of degrees $< n$, then we again obtain a relaxed multiplication for series $f \in \mathbb{M}_1[[z]]$ and $g \in \mathbb{M}_2[[z]]$ at order n of time complexity $\mathcal{O}(\mathsf{M}(n) \log n)$.

Theorem 3. [10] *If $\mathbb{K}$ admits a primitive 2^p-th root of unity for all p, then there exists a relaxed multiplication algorithm of time complexity $\mathsf{R}(n) = \mathcal{O}(n \log n\, \mathrm{e}^{2\sqrt{\log 2 \log\log n}})$. In practice, the existence of a 2^{p+1}-th root of unity with $2^p \geqslant n$ suffices for multiplication up to order n.*

3.2 Relaxed multivariate Laurent series

Let $\mathbb{A}$ be an effective ring. A power series $f \in \mathbb{A}[[z]]$ is said to be *computable* if there is an algorithm which takes $n \in \mathbb{N}$ on input and produces the coefficient f_n on output. We will denote by $\mathbb{A}[[z]]^{\mathrm{com}}$ the set of such series. Then $\mathbb{A}[[z]]^{\mathrm{com}}$ is an effective ring for relaxed addition, subtraction and multiplication.

A computable Laurent series is a formal product $f\, z^k$ with $f \in \mathbb{A}[[z]]^{\mathrm{com}}$ and $k \in \mathbb{Z}$. The set $\mathbb{A}((z))^{\mathrm{com}}$ of such series forms an effective ring for the addition, subtraction and multiplication defined by

$$\begin{aligned} f\, z^k + g\, z^l &= (f\, z^{k-\min(k,l)} + g\, z^{l-\min(k,l)})\, z^{\min(k,l)} \\ f\, z^k - g\, z^l &= (f\, z^{k-\min(k,l)} - g\, z^{l-\min(k,l)})\, z^{\min(k,l)} \\ (f\, z^k)\,(g\, z^l) &= (f\, g)\, z^{k+l}. \end{aligned}$$

If $\mathbb{A}$ is an effective field with an effective zero test, then we may also define an effective division on $\mathbb{A}((z))^{\mathrm{com}}$, but this operation will not be needed in what follows.

Assume now that z is replaced by a finite number of variables $z = (z_1, ..., z_n)$. Then an element of

$$\mathbb{A}((z))^{\mathrm{com}} \;:=\; \mathbb{A}((z_n))^{\mathrm{com}} \cdots ((z_1))^{\mathrm{com}}$$

will also be called a "computable lexicographical Laurent series". Any non zero $f \in \mathbb{A}((z))$ has a natural valuation $v_f = (v_1, ..., v_n) \in \mathbb{Z}^n$, by setting $v_1 = \mathrm{val}_{z_1} f$, $v_2 = \mathrm{val}_{z_2}([z_1^{v_1}]\, f)$, etc. The concept of recursive equations naturally generalizes to the multivariate context. For instance, for an infinitesimal Laurent series $\varepsilon \in \mathbb{A}((z))^{\mathrm{com}}$ (that is, $\varepsilon = f\, z^k$, where $v_f >^{\mathrm{lex}} -k$), the formula

$$g \;=\; 1 + \varepsilon\, g$$

allows us to compute $g = (1 - \varepsilon)^{-1}$ using a single relaxed multiplication in $\mathbb{A}((z))^{\mathrm{com}}$.

Now take $\mathbb{A} = \mathbb{K}[x]$ and consider a polynomial $P \in \mathbb{A}$. Then we define the Laurent polynomial $\hat{P} \in \mathbb{K}[x\, z^{-\lambda}] \subseteq \mathbb{A}((z))^{\mathrm{com}}$ by

$$\hat{P} \;=\; \sum_{i \in \mathbb{N}^n} P_i\, x^i\, z^{-\lambda \cdot i}.$$

Conversely, given $f \in \mathbb{K}[x\, z^{-\lambda}]$, we define $\check{f} \in \mathbb{K}[x]$ by substituting $z_1 = \cdots = z_n = 1$ in f. We will call the transformations $P \mapsto \hat{P}$ and $\hat{P} \mapsto P = \check{\hat{P}}$ *tagging* resp. *untagging*; they provide us with a relaxed mechanism to compute with multivariate polynomials in $\mathbb{K}[x]$, such that the admissible ordering $\leqslant$ on $\mathbb{N}^n$ is respected. For instance, we may compute the relaxed product of two polynomials $P, Q \in \mathbb{K}[x]$ by computing the relaxed product $\hat{P}\hat{Q}$ and substituting $z_1 = \cdots = z_n = 1$ in the result. We notice that tagging is an injective operation which preserves the size of the support.

3.3 Complexity analysis

Assume now that we are given P, $Q \in \mathbb{K}[x]$ and a set $\mathcal{R} \subseteq \mathbb{N}^n$ such that $\mathrm{supp}\,(P\,Q) \subseteq \mathcal{R}$. We assume that $\mathsf{SM}(s)$ is a function such that the (zealous) product PQ can be computed in time $\mathsf{SM}(|\mathcal{R}|)$. We will also assume that $\mathsf{SM}(s)\,/\,s$ is an increasing function of s. In [2, 15], it is shown that we may take $\mathsf{SM}(s) = \mathcal{O}(\mathsf{M}(s) \log s)$.

Let us now study the complexity of sparse relaxed multiplication of P and Q. We will use the classical algorithm for fast univariate relaxed multiplication from [6, 7], of time complexity $\mathsf{R}(s) = \mathcal{O}(\mathsf{M}(s) \log s)$. We will also consider semi-relaxed multiplication as in [8], where one of the arguments $\hat{P}$ or $\hat{Q}$ is completely known in advance and only the other one is computed in a relaxed manner.

Given $X \subseteq \mathbb{N}^n$ and $i \in \{1, ..., n\}$, we will denote

$$\begin{aligned} \delta_i(X) &= \max\{\lambda_i \cdot k : k \in X\} + 1 \\ \delta(X) &= \delta_1(X) \cdots \delta_n(X). \end{aligned}$$

We now have the following:

Theorem 4. *With the above notations, the relaxed product of P and Q can be computed in time*

$$\mathcal{O}(\mathsf{SM}(|\mathcal{R}|)\log\delta(\mathcal{R})).$$

Proof. In order to simplify our exposition, we will rather prove the theorem for a semi-relaxed product of $\hat{P}$ (relaxed) and $\hat{Q}$ (known in advance). As shown in [8], the general case reduces to this special case. We will prove by induction over n that the semi-relaxed product can be computed using at most $3\,\mathsf{SM}(|\mathcal{R}|)\log\delta(\mathcal{R})$ operations in $\mathbb{K}$ if $\mathcal{R}$ is sufficiently large. For $n=0$, we have nothing to do, so assume that $n>0$.

Let us first consider the semi-relaxed product of $\hat{P}$ and $\hat{Q}$ with respect to z_1. Setting $l=\lceil\log_2\delta_1(\mathcal{R})\rceil$, the computation of this product corresponds (see the right-hand side of figure 1) to the computation of $\leqslant 2$ zealous $2^{l-1}\times 2^{l-1}$ products (i.e. 2 products of polynomials of degrees $<2^{l-1}$ in z_1), $\leqslant 4$ zealous $2^{l-2}\times 2^{l-2}$ products, and so on until $\leqslant 2^l$ zealous 1×1 products. We finally need to perform 2^l semi-relaxed 1×1 products of series in $z_2,\ldots,z_n$ only.

More precisely, assume that $\hat{P}$ and $\hat{Q}$ have valuations p resp. q in z_1 and let $\hat{P}_i$ stand for the coefficient of z_1^i in P. We also define

$$\hat{\mathcal{R}} \;=\; \{(a_1,\ldots,a_n,b_1,\ldots,b_n)\in\mathbb{N}^n\times\mathbb{Z}^n\colon (a_1,\ldots,a_n)\in\mathcal{R}\wedge(\forall i, b_i=-\lambda_i\cdot a)\}.$$

Now consider a block size 2^k. For each i, we define

$$\begin{aligned}
\hat{P}_{[i]} &= \hat{P}_{p+2^k i}\,z_1^{p+2^k i}+\cdots+\hat{P}_{p+2^k(i+1)-1}\,z_1^{p+2^k(i+1)-1}\\
\hat{Q}_{[i]} &= \hat{Q}_{q+2^k i}\,z_1^{q+2^k i}+\cdots+\hat{Q}_{q+2^k(i+1)-1}\,z_1^{q+2^k(i+1)-1}\\
\hat{\mathcal{R}}_{[i]} &= \{(a_1,\ldots,a_n,b_1,\ldots,b_n)\in\hat{\mathcal{R}}\colon\\
&\qquad 2^k i\leqslant a_1-p-q\leqslant 2^k(i+1)-1\},
\end{aligned}$$

and notice that the $\hat{\mathcal{R}}_{[i]}$ are pairwise disjoint. In the semi-relaxed multiplication, we have to compute the zealous $2^k\times 2^k$ products $\hat{P}_{[i]}\,\hat{Q}_{[1]}$ for all $i\leqslant\lfloor(\delta_1(\mathcal{R})+1)/2^k\rfloor$. Since

$$\operatorname{supp}\hat{P}_{[i]}\,\hat{Q}_{[1]} \;\subseteq\; \hat{\mathcal{R}}_{[i+1]}\amalg\hat{\mathcal{R}}_{[i+2]},$$

we may compute all these products in time

$$\begin{aligned}
&\mathsf{SM}\big(\big|\hat{\mathcal{R}}_{[1]}\amalg\hat{\mathcal{R}}_{[2]}\big|\big)+\cdots+\mathsf{SM}\big(\big|\hat{\mathcal{R}}_{[2^{l-k}]}\amalg\hat{\mathcal{R}}_{[2^{l-k}+1]}\big|\big)\\
=\;&\big(\big|\hat{\mathcal{R}}_{[1]}\amalg\hat{\mathcal{R}}_{[2]}\big|\big)\,\frac{\mathsf{SM}(|\hat{\mathcal{R}}_{[1]}\amalg\hat{\mathcal{R}}_{[2]}|)}{|\hat{\mathcal{R}}_{[1]}\amalg\hat{\mathcal{R}}_{[2]}|}+\cdots+\\
&\big(\big|\hat{\mathcal{R}}_{[2^{l-k}]}\amalg\hat{\mathcal{R}}_{[2^{l-k}+1]}\big|\big)\,\frac{\mathsf{SM}(|\hat{\mathcal{R}}_{[2^{l-k}]}\amalg\hat{\mathcal{R}}_{[2^{l-k}+1]}|)}{|\hat{\mathcal{R}}_{[2^{l-k}]}\amalg\hat{\mathcal{R}}_{[2^{l-k}+1]}|}\\
\leqslant\;&\big(\big|\hat{\mathcal{R}}_{[1]}\amalg\hat{\mathcal{R}}_{[2]}\big|+\cdots+\big|\hat{\mathcal{R}}_{[2^{l-k}]}\amalg\hat{\mathcal{R}}_{[2^{l-k}+1]}\big|\big)\,\frac{\mathsf{SM}(|\hat{\mathcal{R}}|)}{|\hat{\mathcal{R}}|}\\
\leqslant\;&2\,\mathsf{SM}(|\hat{\mathcal{R}}|)=2\,\mathsf{SM}(|\mathcal{R}|).
\end{aligned}$$

The total time spent in performing all zealous $2^k\times 2^k$ block multiplications with $2^k<2^l$ is therefore bounded by $2\,\mathsf{SM}(|\mathcal{R}|)\log\delta_1(\mathcal{R})$.

Let us next consider the remaining 1×1 semi-relaxed products. If $n = 1$, then these are really scalar products, whence the remaining work can clearly be performed in time $\mathsf{SM}(|\mathcal{R}|) \log \delta_1(\mathcal{R})$ if $\mathcal{R}$ is sufficiently large. If $n > 1$, then for each i, we have

$$\operatorname{supp} \hat{P}_{[i]}\, \hat{Q}_{[0]} \;\subseteq\; \hat{\mathcal{R}}_{[i]}.$$

By the induction hypothesis, we may therefore perform this semi-relaxed product in time $3\,\mathsf{SM}\big(\big|\hat{\mathcal{R}}_{[i]}\big|\big)\,(\log \delta(\mathcal{R}) - \log \delta_1(\mathcal{R}))$. A similar argument as above now yields the bound $3\,\mathsf{SM}(|\mathcal{R}|)\,(\log \delta(\mathcal{R}) - \log \delta_1(\mathcal{R}))$ for performing all 1×1 semi-relaxed block products. The total execution time (which also takes into account the final additions) is therefore bounded by $3\,\mathsf{SM}(|\mathcal{R}|) \log \delta(\mathcal{R})$. This completes the induction. □

Remark 5. In practice, the computation of zealous products of the form $\hat{P}_{[i]}\, \hat{Q}_{[j]}$ is best done in the untagged model, i.e. using the formula

$$\hat{P}_{[i]}\, \hat{Q}_{[j]} \;=\; \widehat{\hat{P}_{[i]}\, \hat{Q}_{[j]}}.$$

Proceeding this way allows us to use any of our preferred algorithms for sparse polynomial multiplication. In particular, we may use [14] or [12].

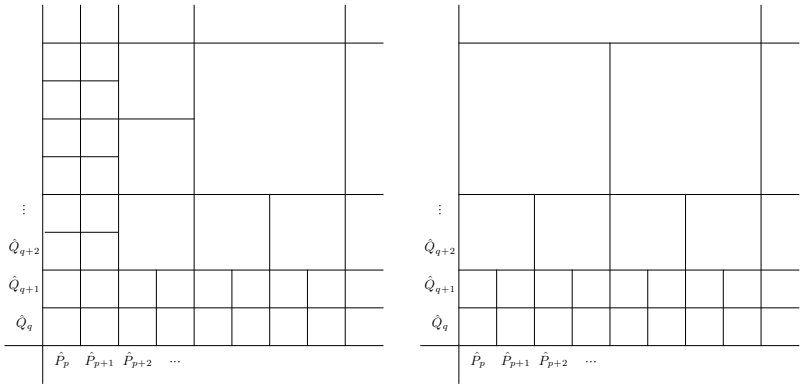

Figure 1. Illustration of a fast relaxed product and a fast semi-relaxed product.

4 Polynomial reduction

4.1 Naive extended reduction

Consider a tuple $B = (B_1, ..., B_b) \in \mathbb{K}[x]^b$. We say that B is *autoreduced* if $B_i \neq 0$ for all i and $l_{B_i} \not\preccurlyeq l_{B_j}$ and $l_{B_j} \not\preccurlyeq l_{B_i}$ for all $i \neq j$. Given such a tuple B and an arbitrary polynomial $A \in \mathbb{K}[x]$, we say that A is *reduced* with respect to B if $l_{B_i} \not\preccurlyeq k$ for all i and $k \in \operatorname{supp} A$. An *extended reduction* of A with respect to B is a tuple $(Q_1, ..., Q_b, R)$ with

$$A \;=\; Q_1\, B_1 + \cdots + Q_b\, B_b + R, \tag{3}$$

such that R is reduced with respect to B. The naive algorithm extended-reduce below computes an extended reduction of A.

Algorithm extended-reduce
Input: $A \in \mathbb{K}[x]$ and an autoreduced tuple $B \in \mathbb{K}[x]^b$
Output: an extended reduction of A with respect to B

Start with $Q := (0, ..., 0)$ and $R := A$
While R is not reduced with respect to B do
  Let i be minimal and such that $l_{B_i} \preccurlyeq k$ for some $k \in \operatorname{supp} R$
  Let $k \in \operatorname{supp} R$ be maximal with $l_{B_i} \preccurlyeq k$
  Set $Q_i := Q_i + (R_k / c_{B_i})\, x^{k - l_{B_i}}$ and $R := R - (R_k / c_{B_i})\, x^{k - l_{B_i}} B_i$
Return $(Q_1, ..., Q_b, R)$

Remark 6. Although an extended reduction is usually not unique, the one computed by extended-reduce is uniquely determined by the fact that, in our main loop, we take i minimal with $l_{B_i} \preccurlyeq k$ for some $k \in \operatorname{supp} R$. This particular extended reduction is also characterized by the fact that

$$\operatorname{supp} Q_i + l_{B_i} \;\subseteq\; \operatorname{Fin}(\{l_{B_i}\}) \setminus \operatorname{Fin}(\{l_{B_1}, ..., l_{B_{i-1}}\})$$

for each i.

In order to compute $Q_1, ..., Q_b$ and R in a relaxed manner, upper bounds

$$\begin{aligned} \operatorname{supp} Q_i &\subseteq \mathcal{Q}_i \\ \operatorname{supp} Q_i B_i &\subseteq \mathcal{Q}_i + \operatorname{supp} B_i \\ \operatorname{supp} R &\subseteq \mathcal{R} \end{aligned}$$

need to be known beforehand. These upper bounds are easily computed as a function of $\mathcal{A} = \operatorname{supp} A$, $\mathcal{B}_1 = \operatorname{supp} B_1$, ..., $\mathcal{B}_b = \operatorname{supp} B_b$ by the variant supp-extended-reduce of extended-reduce below. We recall from the end of the introduction that we do not take into account the cost of this computation in our complexity analysis. In reality, the execution time of supp-extended-reduce is similar to the one of extended-reduce, except that potentially expensive operations in $\mathbb{K}$ are replaced by boolean operations of unit cost. We also recall that support bounds can often be obtained by other means for specific problems.

Algorithm supp-extended-reduce
Input: subsets $\mathcal{A}$ and $\mathcal{B}_1, ..., \mathcal{B}_b$ of $\mathbb{N}^n$ as above
Output: subsets $\mathcal{Q}_1, ..., \mathcal{Q}_b$ and $\mathcal{R}$ of $\mathbb{N}^n$ as above

Start with $\mathcal{Q} := (\varnothing, ..., \varnothing)$ and $\mathcal{R} := \mathcal{A}$
While $\mathcal{R} \cap \operatorname{Fin}(\{\max \mathcal{B}_1, ..., \max \mathcal{B}_b\}) \neq \varnothing$ do
  Let i be minimal with $l_{\max \mathcal{B}_i} \preccurlyeq k$ for some $k \in \mathcal{R}$
  Let $k \in \mathcal{R}$ be maximal with $l_{\max \mathcal{B}_i} \preccurlyeq k$
  Set $\mathcal{Q}_i := \mathcal{Q}_i \cup \{k - \max \mathcal{B}_i\}$ and
    $\mathcal{R} := \mathcal{R} \cup (\mathcal{B}_i + (k - \max \mathcal{B}_i)) \setminus \{k\}$
Return $(\mathcal{Q}_1, ..., \mathcal{Q}_b, \mathcal{R})$

4.2 Relaxed extended reduction

Using the relaxed multiplication from section 3, we are now in a position to replace the algorithm extended-reduce by a new algorithm, which directly computes $Q_1, ..., Q_b, R$ using the equation (3). In order to do this, we still have to put it in a recursive form which is suitable for relaxed resolution.

Denoting by e_i the i-th canonical basis vector of $\mathbb{K}[x]^{b+1}$, we first define an operator $\Phi: x_1^{\mathbb{N}} \cdots x_n^{\mathbb{N}} \to \mathbb{K}[x]^{b+1}$ by

$$\Phi(x^k) \;=\; \begin{cases} c_{B_i}^{-1}\, x^{k-l_{B_i}}\, e_i & \text{if } k \in \operatorname{Fin}(\{l_{B_i}, ..., l_{B_b}\}) \text{ and} \\ & i \text{ is minimal with } l_{B_i} \preccurlyeq k \\ e_{b+1}\, x^k & \text{otherwise} \end{cases}$$

By linearity, this operator extends to $\mathbb{K}[x]$

$$\Phi(P) \;=\; \sum_{i \in \operatorname{supp} P} P_i\, \Phi(x^i).$$

In particular, $\Phi(c_A\; x^{l_A})$ yields the "leading term" of the extended reduction $(Q_1, ..., Q_b, R)$. We also denote by $\hat{\Phi}$ the corresponding operator from $\mathbb{K}[x\, z^{-\lambda}]$ to $\mathbb{K}[x\, z^{-\lambda}]^{b+1}$ which sends $\hat{P}$ to $\widehat{\Phi(P)}$.

Now let $B_i^* = B_i - c_{B_i}\, x^{l_{B_i}}$ for each i. Then

$$(Q_i\, B_i)_k \;=\; (Q_i\, B_i^*)_k + (Q_i)_{k-l_{B_i}}\, c_{B_i}$$

for each $i \in \{1, ..., b\}$ and $k \in \mathbb{N}^n$. The equation

$$(Q_1\, B_1 + \cdots + Q_b\, B_b + R)_k \;=\; A_k$$

can thus be rewritten as

$$\begin{aligned} & (Q_1)_{k-l_{B_1}}\, c_{B_1} + ... + (Q_i)_{k-l_{B_b}}\, c_{B_b} \\ =\; & (A - Q_1\, B_1^* - \cdots - Q_b\, B_b^*)_k \end{aligned}$$

Using the operator Φ this equation can be rewritten in a more compact form as

$$(Q_1, ..., Q_b, R) \;=\; \Phi(A - Q_1\, B_1^* - \cdots - Q_b\, B_b^*).$$

The tagged counterpart

$$\big(\hat{Q}_1, ..., \hat{Q}_b, \hat{R}\big) \;=\; \hat{\Phi}\big(\hat{A} - \hat{Q}_1\, \hat{B}_1^* - \cdots - \hat{Q}_b\, \hat{B}_b^*\big)$$

is recursive, whence the extended reduction can be computed using b multivariate relaxed multiplications $\hat{Q}_1\, \hat{B}_1^*, ..., \hat{Q}_b\, \hat{B}_b^*$. With $\mathcal{A}, \mathcal{B}_i, \mathcal{Q}_i$ and $\mathcal{R}$ as in the previous section, theorem 4 therefore implies:

Theorem 7. *We may compute the extended reduction of A with respect to B in time*

$$\begin{aligned} & \mathcal{O}(\mathsf{SM}(|\mathcal{B}_1 + \mathcal{Q}_1|) \log \delta(\mathcal{B}_1 + \mathcal{Q}_1) + \cdots + \\ & \quad \mathsf{SM}(|\mathcal{B}_b + \mathcal{Q}_b|) \log \delta(\mathcal{B}_b + \mathcal{Q}_b) + |\mathcal{R}|). \end{aligned}$$

Remark 8. Following remark 2, we also notice that A, the Q_i and R may be replaced by vectors of polynomials in $\mathbb{K}[x]^m$ (regarded as polynomials with coefficients in $\mathbb{K}^m$), in the case that several polynomials need to be reduced simultaneously.

Bibliography

[1] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 301–309, New York, NY, USA, 1988. ACM Press.

[2] J. Canny, E. Kaltofen, and Y. Lakshman. Solving systems of non-linear polynomial equations faster. In *Proc. ISSAC '89*, pages 121–128, Portland, Oregon, A.C.M., New York, 1989. ACM Press.

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

[4] M. J. Fischer and L. J. Stockmeyer. Fast on-line integer multiplication. *Proc. 5th ACM Symposium on Theory of Computing*, 9:67–72, 1974.

[5] D. Y. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pages 166–172, 1987.

[6] J. van der Hoeven. Lazy multiplication of formal power series. In W. W. Küchlin, editor, *Proc. ISSAC '97*, pages 17–20, Maui, Hawaii, July 1997.

[7] J. van der Hoeven. Relax, but don't be too lazy. *JSC*, 34:479–542, 2002.

[8] J. van der Hoeven. Relaxed multiplication using the middle product. In Manuel Bronstein, editor, *Proc. ISSAC '03*, pages 143–147, Philadelphia, USA, August 2003.

[9] 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.

[10] J. van der Hoeven. New algorithms for relaxed multiplication. *JSC*, 42(8):792–802, 2007.

[11] J. van der Hoeven. From implicit to recursive equations. Technical report, HAL, 2011. http://hal.archives-ouvertes.fr/hal-00583125.

[12] J. van der Hoeven and G. Lecerf. On the complexity of blockwise polynomial multiplication. In *Proc. ISSAC '12*, pages 211–218, Grenoble, France, July 2012.

[13] J. van der Hoeven and G. Lecerf. On the bit-complexity of sparse polynomial multiplication. *JSC*, 50:227–254, 2013.

[14] J. van der Hoeven and É. Schost. Multi-point evaluation in higher dimensions. *AAECC*, 24(1):37–52, 2013.

[15] E. Kaltofen and Y. N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In *ISSAC '88: Proceedings of the international symposium on Symbolic and algebraic computation*, pages 467–474. Springer Verlag, 1988.

[16] L. Robbiano. Term orderings on the polynominal ring. In *European Conference on Computer Algebra (2)*, pages 513–517, 1985.