Fast Gröbner basis computation and polynomial reduction for generic bivariate ideals

Joris van der Hoeven, Robin Larrieu

Fast Gröbner basis computation and polynomial reduction for generic bivariate ideals

JORIS VAN DER HOEVEN[a], ROBIN LARRIEU[b]

Laboratoire d'informatique de l'École polytechnique
LIX, UMR 7161 CNRS
Campus de l'École polytechnique
1, rue Honoré d'Estienne d'Orves
Bâtiment Alan Turing, CS35003
91120 Palaiseau, France

a. *Email:* vdhoeven@lix.polytechnique.fr
b. *Email:* larrieu@lix.polytechnique.fr

Preliminary version of January 28, 2019

Let $A, B \in \mathbb{K}[X, Y]$ be two bivariate polynomials over an effective field $\mathbb{K}$, and let G be the reduced Gröbner basis of the ideal $I := \langle A, B \rangle$ generated by A and B with respect to the usual degree lexicographic order. Assuming A and B sufficiently generic, we design a quasi-optimal algorithm for the reduction of $P \in \mathbb{K}[X, Y]$ modulo G, where "quasi-optimal" is meant in terms of the size of the input A, B, P. Immediate applications are an ideal membership test and a multiplication algorithm for the quotient algebra $\mathbb{A} := \mathbb{K}[X, Y] / \langle A, B \rangle$, both in quasi-linear time. Moreover, we show that G itself can be computed in quasi-linear time with respect to the output size.

KEYWORDS: polynomial reduction; Gröbner basis; complexity; algorithm

A.M.S. SUBJECT CLASSIFICATION: 13P10

1. INTRODUCTION

Gröbner bases are a powerful tool for solving systems of polynomial equations, or to compute modulo polynomial ideals. The research area dedicated to their computation is very active, and there is an abundant literature on efficient algorithms for this task. See for example [5, 6, 8] and references therein. Although this problem requires exponential space in the worst case [22], it is in fact tractable for many practical instances. For example, computer algebra systems often implement one of Faugère's F4 or F5 algorithms [5, 6] that are very efficient if the system has sufficient regularity. A polynomial complexity bound for the F5 algorithm was established in [1], when counting the number of field operations in terms of the expected output size.

The F4 and F5 algorithms and all other currently known fast algorithms for Gröbner basis computations rely on linear algebra, and it may seem surprising that quasi-optimal FFT-based polynomial arithmetic is not used in this area. This can be seen as an illustration of how difficult it is to compute Gröbner bases, but there is another explanation: traditionally, Gröbner basis algorithms consider a large number of variables and the degree of the generating polynomials is kept small; but fast polynomial arithmetic works best in the opposite regime (a fixed number of variables and large degrees). Even in this setting, it is not clear how to use FFT techniques for Gröbner basis computation.

Operation	Our paper	Previous best
Deglex Gröbner basis G	$O(\mathsf{R}(m^2) + \mathsf{R}(nm)\, n \log n) = \tilde{O}(\lvert A, B, G\rvert)$	$O(n^2\, \mathsf{R}(nm))$
Structure of $\mathbb{K}[X,Y]/\langle A, B\rangle$	$O(\mathsf{R}(m^2) + \mathsf{M}(nm) \log n) = \tilde{O}(\lvert A, B\rvert)$	$O(n^2\, \mathsf{R}(nm))$
Reduction of P with $\deg P = d$	$O(\mathsf{R}(d^2) + \mathsf{R}(nm) \log n) = \tilde{O}(\lvert P\rvert + D)$	$O(\mathsf{R}(d^2) + n\, \mathsf{R}(nm))$
Multiplication in $\mathbb{K}[X,Y]/\langle A, B\rangle$	$O(\mathsf{R}(nm) \log n) = \tilde{O}(D)$	$O(n\, \mathsf{R}(nm))$

Table 1. Asymptotic complexities for various problems on the ideal $\langle A, B\rangle$, assuming $n := \deg A \leqslant m := \deg B$. The notations M and R represent cost functions for multiplication and relaxed multiplication respectively (see section 3). In the last two rows, for the computation of normal forms and multiplication in $\mathbb{K}[X,Y]/\langle A, B\rangle$, we assume that the structure of $\mathbb{K}[X,Y]/\langle A, B\rangle$ has been precomputed using the algorithm from the second row. Notice that the algorithm for fast reduction from the third line also yields an ideal membership test and an algorithm to compute normal forms modulo $\langle A, B\rangle$.

As a first step, one may consider related problems, such as the reduction of multivariate polynomials. Given a Gröbner basis $G := (G_0, \ldots, G_n)$, we wish to reduce a polynomial P with respect to G, that is find $(Q_0, \ldots, Q_n, R)$ such that $P = Q_0 G_0 + \cdots + Q_n G_n + R$ and R cannot be further reduced. It was shown in [17] that reduction can be done in quasi-linear time with respect to the size of the equation $P = Q_0 G_0 + \cdots + Q_n G_n + R$. Using standard Gröbner basis techniques (see sections 2 and 6.2), this leads to the complexities announced in the last column of Table 1 for various other operations.

Unfortunately, the equation $P = Q_0 G_0 + \cdots + Q_n G_n + R$ is in general much larger than the intrinsic complexity of the problem, given by the size of P and the degree D of the ideal (which is linked to the size of the generating polynomials). Recent work in the bivariate setting [18] gave an asymptotically optimal reduction algorithm for a particular class of Gröbner bases. This algorithm relies on a terse representation of $G := (G_0, \ldots, G_n)$ in $\tilde{O}(D)$ space, where $\tilde{O}$ stands for the "soft Oh" notation (that hides poly-logarithmic factors) [12, Section 25.7]. Assuming that this representation has been precomputed, the extended reduction can be performed in time $\tilde{O}(\lvert P\rvert + D)$, instead of the previous $\tilde{O}(\lvert P\rvert + \lvert G\rvert)$ where $\lvert G\rvert = \Theta(nD)$.

Instead of making regularity assumptions on the Gröbner basis itself, one may focus on the generating polynomials. If the ideal is defined by generic polynomials given in total degree, then the Gröbner basis presents a particular structure, as studied for example in [11, 10, 23]. This situation is often used as a benchmark for polynomial system solving: see the PoSSo problem [7]. In this paper, we restrict ourselves to the bivariate case, as studied for example in [21].

In what follows, $A, B \in \mathbb{K}[X,Y]$ are generic polynomials of degree n, m respectively. We denote by $\langle A, B\rangle$ the ideal they generate, and we consider its Gröbner basis with respect to the graded lexicographic order. Under these very particular assumptions, it turns out that determining a Gröbner basis essentially boils down to a univariate gcd computation. More precisely, the computation of the gcd of the two leading diagonals gives us the necessary information to define and compute a *concise representation* for a Gröbner basis of $\langle A, B\rangle$. We show that the computation of a Gröbner basis in this representation can be done in quasi-linear time $\tilde{O}(D)$ and the same holds for the reduction of a polynomial with respect to the Gröbner basis. Combining these two algorithms, we obtain an ideal membership test $P \in^? \langle A, B\rangle$ in quasi-linear time $\tilde{O}(\lvert P\rvert + D)$. Similarly, multiplications in the quotient algebra $\mathbb{A} := \mathbb{K}[X,Y]/\langle A, B\rangle$ can be done with the same complexity. Finally, we show that the reduced Gröbner basis in the classical sense can be computed in quasi-linear time with respect to the output size $\Theta(D \min(n, m))$. Our complexity results are summarized in Table 1.

The concept of a concise representation of Gröbner bases bears some similarities with the terse representations from [18]. Nevertheless, under more restrictive assumptions, the results in the present paper are considerably stronger: contrary to [18], our complexity bounds do not rely on potentially expensive precomputations. In fact, we show how to compute a Gröbner basis for $\langle A, B\rangle$ in quasi-linear time. See section 2.5 for more details on the differences with [18].

Structure of the paper. Section 2 gives a general overview, with the hypotheses and the main ideas of our algorithm. In section 3, we recall the complexities for some fundamental operations on polynomials. The structure and the computation of the concise representation are presented in section 4, then the reduction algorithm is detailed in section 5. In section 6, we give the algorithms for the ideal membership, multiplication in $\mathbb{K}[X,Y]/\langle A,B\rangle$ and the reduced Gröbner basis. For convenience of the reader, sections 5 and 6 actually present simplified variants of our algorithms with complexities that are slightly weaker than those in Table 1. In section 7, we detail the technical changes that are needed to obtain the announced results in full generality.

Notations and terminology. We assume that the reader is familiar with the theory of Gröbner bases and refer to [12, 2] for basic expositions. We denote the set of *monomials* in r variables by $\mathcal{M} := \{X_1^{i_1}\cdots X_r^{i_r} : i_1,\ldots,i_r \in \mathbb{N}\}$. A *monomial ordering* $\prec$ on $\mathcal{M}$ is a total ordering that is compatible with multiplication. Given a polynomial in r variables $P = \sum_{M\in\mathcal{M}} P_M M \in \mathbb{K}[X_1,\ldots,X_r]$, its *support* $\operatorname{supp} P$ is the finite set of monomials $M\in\mathcal{M}$ with $P_M \neq 0$. If $P\neq 0$, then $\operatorname{supp} P$ admits a maximal element for $\prec$ that is called its *leading monomial* and that we denote by $\operatorname{lm}(P)$. If $M\in\operatorname{supp} P$, then we say that $P_M M$ is a *term* in P. Given a tuple $A = (A_0,\ldots,A_n)$ of polynomials in $\mathbb{K}[X_1,\ldots,X_r]$, we say that P is *reduced* with respect to A if $\operatorname{supp} P$ contains no monomial that is a multiple of the leading monomial of one of the A_i.

Unless stated otherwise, we will always work in the bivariate setting when $r=2$, and use X and Y as our main indeterminates instead of X_1 and X_2. In particular, $\mathcal{M} := \{X^a Y^b : a,b\in\mathbb{N}\}$. Moreover, we only consider the usual degree lexicographic order with $X\prec Y$, that is

$$X^a Y^b \prec X^u Y^v \Leftrightarrow a+b<u+v \text{ or } (a+b=u+v \text{ and } b<v).$$

Acknowledgements. We thank Vincent Neiger for a remark that simplified Algorithm 6. We also thank the anonymous referees for helpful comments and suggestions.

2. PRESENTATION OF THE SETTING

2.1. Reduced Gröbner bases

We consider an ideal $I\subset\mathbb{K}[X,Y]$ generated by two generic polynomials A,B of total degree n,m and we assume $n\leqslant m$. Here the adjective "generic" should be understood as "no accidental cancellation occurs during the computation". This is typically the case if A,B are chosen at random: assuming that $\mathbb{K}$ has sufficiently many elements, the probability of an accidental cancellation is small. In this generic bivariate setting, a clever application of Buchberger's algorithm [3] gives the reduced Gröbner basis $G^{\mathrm{red}} = (G_0^{\mathrm{red}},\ldots,G_n^{\mathrm{red}})$ of I with respect to $\prec$ as follows:

- Set $G_1^{\mathrm{red}} := B \operatorname{rem} A$ and $G_0^{\mathrm{red}} := A \operatorname{rem} G_1^{\mathrm{red}}$.
- $G_i^{\mathrm{red}} := \operatorname{Spol}(G_{i-2}^{\mathrm{red}}, G_{i-1}^{\mathrm{red}}) \operatorname{rem} (G_0^{\mathrm{red}},\ldots,G_{i-1}^{\mathrm{red}})$ for $i=2,\ldots,n$.
- For all $i=0,\ldots,n$, divide G_i^{red} by its leading coefficient to make it monic.

It is well known [11] that the Gröbner stair has steps of height 1, so the algorithm stops when $i=n=\deg A$, or equivalently when the leading term of G_i^{red} is a power of X. It can also be checked that the width of each step is 2, except for the first one that has width $n-m+1$. In other words,

the leading monomials $\mathrm{lm}(G_i^{\mathrm{red}})$ of the G_i^{red} are given by

$$\mathrm{lm}(G_0^{\mathrm{red}}) = Y^n \tag{1}$$
$$\mathrm{lm}(G_i^{\mathrm{red}}) = X^{m-n-1+2i} Y^{n-i}, \qquad i=1,\ldots,n. \tag{2}$$

There are $n\,m$ monomials under the stairs; since the Bézout bound is reached for generic ideals, this ensures that G^{red} is indeed a reduced Gröbner basis.

2.2. From Euclidean division to Gröbner bases

Let us now show how to construct another (non-reduced) Gröbner basis $G=(G_0,\ldots,G_n)$ with even simpler recurrence relations. These recurrence relations will be one ingredient of our fast reduction algorithm.

DEFINITION 1. *For a polynomial $P=\sum P_{i,j}X^iY^j\in\mathbb{K}[X,Y]$ of total degree d, we define its dominant diagonal* $\mathrm{Diag}(P)\in\mathbb{K}[Z]$ *by* $\mathrm{Diag}(P)=\sum_{j\leqslant d}P_{d-j,j}Z^j$.

We have the trivial properties that $\mathrm{Diag}(XP)=\mathrm{Diag}(P)$ and $\mathrm{Diag}(YP)=Z\,\mathrm{Diag}(P)$. For generic A and B, the diagonals $\mathrm{Diag}(A)$ and $\mathrm{Diag}(B)$ are also generic. Applying the Euclidean algorithm to these diagonals, it follows that the successive remainders follow a "normal sequence", i.e. their degrees decrease by exactly one at each step.

Now consider the sequence $G_0,\ldots,G_n$ with $n=\deg A$ defined by

$$G_0 := A \tag{3}$$
$$G_1 := B \operatorname{rem} A \tag{4}$$
$$G_i := X^{d_i}G_{i-2}-(u_iY+v_iX)\,G_{i-1}, \qquad i=2,\ldots,n, \tag{5}$$

where

$$u_iZ+v_i := \mathrm{Diag}(G_{i-2}) \operatorname{quo} \mathrm{Diag}(G_{i-1}), \qquad d_i := \begin{cases} m-n+1 & \text{if } i=2 \\ 2 & \text{if } i>2 \end{cases}.$$

Let us first notice that the term $X^{d_i}G_{i-2}-u_iYG_{i-1}$ corresponds to the S-polynomial of G_{i-2} and G_{i-1}, as in the classical Buchberger algorithm. Setting $D_i:=\mathrm{Diag}(G_i)$ for $i=0,\ldots,n$, we next observe that

$$D_1 = \mathrm{Diag}(B) \operatorname{rem} \mathrm{Diag}(A)$$
$$D_i = D_{i-2} \operatorname{rem} D_{i-1}, \qquad i=2,\ldots,n,$$

so the diagonals are the successive remainders in the Euclidean algorithm and the corresponding quotients indeed all have degree 1. By induction on i, we deduce:

LEMMA 2. *The G_i as in (3–5) have the same leading monomials (1–2) as the G_i^{red}, so $G:=(G_0,\ldots,G_n)$ is a Gröbner basis of $\langle A,B\rangle$ with respect to $\prec$.*

COROLLARY 3. *Any $P\in\mathbb{K}[X,Y]$ has the same normal form with respect to G and G^{red}.*

Remark 4. The genericity assumptions on A and B can also be made more precise now: on the one hand, we need that $\deg D_i=n-i$ for $i=0,\ldots,n$ and $\deg G_i=m+i-1$ for $i=1,\ldots,n$. On the other hand, we need $X^2G_{n-1}-(u_nY+v_nX)\,G_n$ to reduce to zero with respect to G, where $u_nZ+v_n:=D_{n-1}\operatorname{quo}D_n$. This in particular provides us with an *a posteriori* sanity check for ensuring that the genericity assumptions are indeed satisfied.

2.3. Examples

Example 5. Consider $A:=Y+aX+b$ and $B:=Y^4+\cdots$, then we have $G_0=Y+aX+b$ and $G_1=B(X,-aX-b)=cX^4+\cdots$ for some c. The sequence stops here since $n:=\deg A=1$. Notice that $(G_0,G_1/c)$ is actually the reduced Gröbner basis in this case.

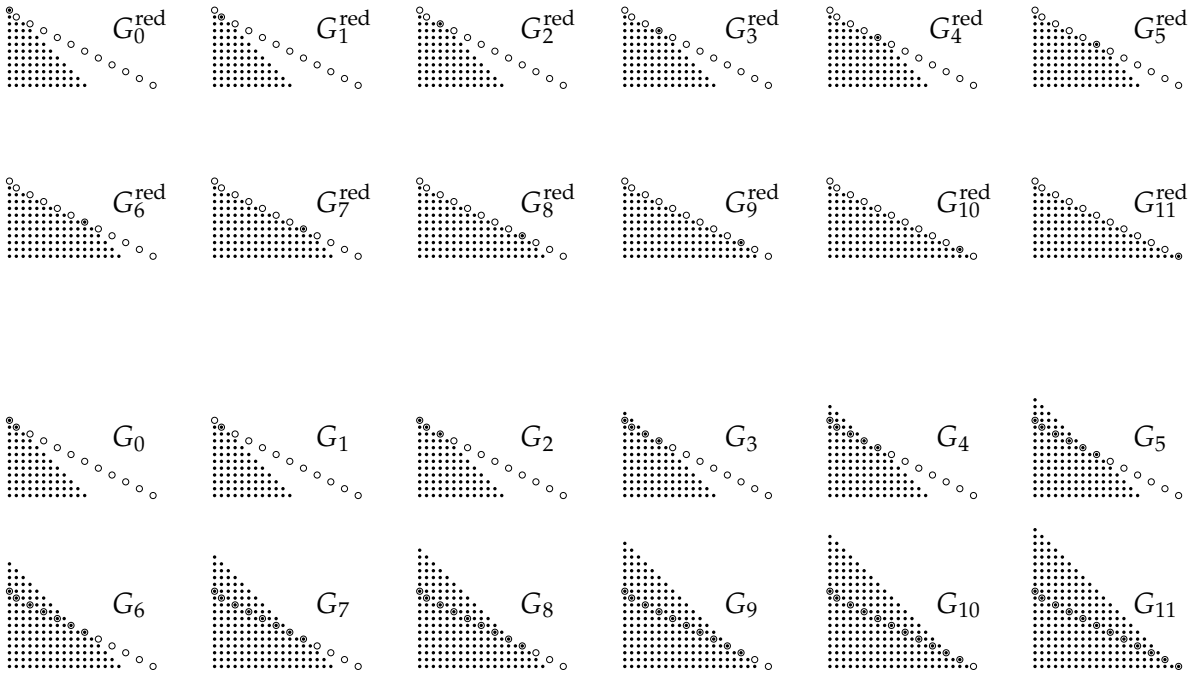

Figure 1. The difference between the reduced Gröbner basis (above) and the non-reduced one obtained as in (3–5) (below).

Example 6. Consider $A = Y^4 - 3XY^3 + X^2Y^2 - 3X^3Y + 5X^4 + 3Y^3 + 2XY^2 - 3X^2Y - 4X^3$ and $B = Y^4 + 2XY^3 - 3X^2Y^2 + 4X^3Y - 3X^4 - 5Y^3 - 3XY^2 + 5X^2Y - 5X^3$ over the field $\mathbb{Z}/11\mathbb{Z}$. The reader's favourite computer algebra system gives the following reduced Gröbner basis:

$$\begin{aligned}
G_0^{\mathrm{red}} &= Y^4 + 3X^2Y^2 - X^3Y - 2X^4 - 4Y^3 - XY^2 + 4X^2Y + 2X^3,\\
G_1^{\mathrm{red}} &= XY^3 - 3X^2Y^2 - 3X^3Y + 5X^4 + 5Y^3 - XY^2 - 5X^2Y + 2X^3,\\
G_2^{\mathrm{red}} &= X^3Y^2 - 2X^4Y + 4X^5 + 3X^2Y^2 - 5X^3Y - 3X^4 + Y^3 - 3XY^2 - X^2Y - 5X^3,\\
G_3^{\mathrm{red}} &= X^5Y - X^6 - 4X^4Y - 4X^5 - 5X^2Y^2 - 3X^3Y - 4X^4 + Y^3 + 5XY^2 - X^2Y + X^3,\\
G_4^{\mathrm{red}} &= X^7 - 5X^6 - 5X^4Y + 2X^5 + 5X^2Y^2 - X^3Y - X^4 - 3Y^3 - 4XY^2 + 3X^2Y - 3X^3.
\end{aligned}$$

Computing the G_i as in (3–5), we obtain

$$\begin{aligned}
G_0 &= Y^4 - 3XY^3 + X^2Y^2 - 3X^3Y + 5X^4 + 3Y^3 + 2XY^2 - 3X^2Y - 4X^3,\\
G_1 &= 5XY^3 - 4X^2Y^2 - 4X^3Y + 3X^4 + 3Y^3 - 5XY^2 - 3X^2Y - X^3,\\
G_2 &= 4X^3Y^2 + 3X^4Y + 5X^5 - 5Y^4 + 4XY^3 - 4X^2Y^2 - 5X^3Y - 4X^4,\\
G_3 &= -X^5Y + X^6 - 2Y^5 - 3XY^4 + 2X^2Y^3 - X^3Y^2 + 4X^4Y + 5X^5,\\
G_4 &= X^7 + 3Y^6 - 4XY^5 + 4X^2Y^4 + 3X^3Y^3 + 5X^4Y^2 - X^5Y - 2X^6.
\end{aligned}$$

Notice that G_i and G_i^{red} have the same leading monomial, so $G := (G_0, \ldots, G_4)$ is a Gröbner basis as well. However, it is not reduced: G_4 contains the term $3Y^6$, which is divisible by the leading monomial of $G_0 = Y^4 + \cdots$.

Example 7. Figure 1 shows a schematic representation of the behavior when A and B have degree 11; this is the example that we will use in the rest of the paper. The black dots (•) represent the supports of the polynomials, while the larger white dots (◯) give the shapes of the Gröbner stairs. Notice again that G_i and G_i^{red} have the same leading monomial.

Remark 8. The second example already involves a lot of coefficients, even for the modest degree 4. Such low degrees are not sufficient to convey the main ideas of our method. For this reason, we are unable to produce meaningful examples for which all coefficients are given explicitly. We therefore chose to present our running example (Example 7) in a more schematic form.

2.4. Key ingredients for our algorithms

The main obstruction for a quasi-linear reduction algorithm is the size of the Gröbner basis: if both A and B have degree n, then the Gröbner basis requires $\Theta(n^3)$ space, whereas the intrinsic complexity of the problem is given by the size of A and B, that is only $\Theta(n^2)$. To achieve $\tilde{O}(n^2)$ complexity bounds, we cannot afford to explicitly store and manipulate the entire Gröbner basis, so we must find a way to compress the relevant information. Our technique is based on the following three ingredients (more details are given in section 4):

- The first idea is to control the degrees of the quotients using a so-called *dichotomic selection strategy*. When reducing a multivariate polynomial, its terms are reduced one after the other against some basis element. There may be several possibilities to choose this basis element and in this case, the usual strategy is to select the first valid choice; however, this method gives quotients with roughly the same degree. On the contrary, with the strategy detailed in section 4.1, most quotients are very small and only a few have a potentially high degree.
- Secondly, the basis elements are *truncated to an appropriate precision* according to the degree bounds established before. The goal is to keep as little information as possible and still be able to perform the operation efficiently. Roughly speaking, if we know that $\deg Q_i \leqslant \delta$, then Q_i depends only on the terms of G_i with degree at least $\deg G_i - \delta$ and the remaining terms may be discarded. This constitutes the first part of the *concise representation* for a Gröbner basis, defined more precisely in section 4.2.
- The third idea is to *rewrite the equation* $P = Q_0 G_0 + \cdots + Q_n G_n + R$ to ensure that the final result is correct despite the aforementioned truncations. This is done by keeping track of the relations that exist among the G_i: recall that G contains far more coefficients than A and B, so there must be some redundancy. A well chosen, sufficiently small subset of these relations constitute the second part of the concise representation: see section 4.2.

2.5. Comparison with terse representations of vanilla Gröbner bases

As mentioned in the introduction, it was already shown in [18] that reduction is possible in quasi-linear time with respect to a certain class of so-called *vanilla Gröbner bases*, through the use of so-called *terse representations* of such Gröbner bases. While many of the techniques are similar, there are also fundamental differences between the two settings. For the interested reader who is familiar with [18], let us briefly outline these differences. Notice that this subsection can safely be skipped.

The first difference concerns the kind of genericity assumptions that we make. In the case of vanilla Gröbner bases, the following assumptions are made *on the basis itself*:

1. The shape of the Gröbner stairs must be regular: the monomials below it are the minimal elements with respect to the monomial ordering to be considered.
2. Suitable relations exist among the basis elements (similar to the ones mentioned in the third ingredient in section 2.4).

In the present paper, the genericity hypotheses are made on the two *generators of the ideal*. Whereas various weighted degree orders can be considered for vanilla Gröbner bases, we crucially require the degree lexicographic order in this paper.

The vanilla situation appears to be generic: for various types of generators and for various term orders, if two generators are picked at random, then the Gröbner basis is vanilla. However, if two random generators are given for the usual total degree, then the deglex basis is not vanilla, because the shape of the stairs does not match. This means the results from [18] do not apply in the setting of this paper, and conversely the results of this paper do not apply for vanilla Gröbner bases.

The concise representation from section 4.2 is also somewhat different from its terse analogue for vanilla bases. Whereas we give an efficient quasi-linear algorithm for the computation of concise representations, terse representations are only assumed to exist (by the vanilla hypothesis), so we rather precompute them (using some algorithm that may not run in quasi-linear time). For this reason, the algorithms in this paper run in quasi-linear time, whereas the reduction algorithm from [18] only runs in quasi-linear time *modulo* suitable precomputations.

3. ALGORITHMIC PREREQUISITES

In this section, we quickly review some basic complexities for fundamental operations on polynomials over a field $\mathbb{K}$. Notice that results presented in this section are not specific to the bivariate case. Running times will always be measured in terms of the required number of field operations in $\mathbb{K}$.

3.1. Polynomial multiplication

We denote by $\mathsf{M}(d)$ the cost of multiplying two dense univariate polynomials of degree d in $\mathbb{K}[X]$. Over general fields $\mathbb{K}$, one may take [25, 24, 4]

$$\mathsf{M}(d) = O(d \log d \log\log d).$$

In the case of fields of positive characteristic, one may even take [14, 15]

$$\begin{aligned} \mathsf{M}(d) &= O(d \log d\, 4^{\log^* d}) \\ \log^* d &= \min\{k \in \mathbb{N} : (\log \circ \overset{k\times}{\cdots} \circ \log)(d) \leqslant 1\}. \end{aligned}$$

We make the customary assumptions that $\mathsf{M}(d)/d$ is increasing and that $\mathsf{M}(2d) = O(\mathsf{M}(d))$, with the usual implications, such as $\mathsf{M}(d) + \mathsf{M}(e) \leqslant \mathsf{M}(d+e)$.

For multivariate polynomials, the cost of multiplication depends on the geometry of the support. The classical example involves dense "block" polynomials in $\mathbb{K}[X_1, \ldots, X_r]$ of degree $< d_i$ in each variable X_i. This case can be reduced to multiplication of univariate polynomials of degree $< 2^{r-1} d_1 \cdots d_r$ using the well known technique of Kronecker substitution [12, Section 8.4]. More generally, for polynomials such that the support of the product is included in an initial segment with d elements, it is possible to compute the product in time $O(\mathsf{M}(d))$ [19]. Here an initial segment of $\mathcal{M}$ is a (finite) subset $\mathcal{S}$ such that for any monomial $M \in \mathcal{S}$, all its divisors are again in $\mathcal{S}$.

For the purpose of this paper, we need to consider dense polynomials P in $\mathbb{K}[X, Y]$ whose supports are contained in sets of the form $S_{l,h} := \{M \in \mathcal{M} : l \leqslant \deg M \leqslant h\}$. Modulo the change of variables $X^a Y^b \to T^{h-a-b} U^b$, such a polynomial can be rewritten as $P(X, Y) = \tilde{P}(T, U)$, where the support of $\tilde{P}$ is an initial segment with the same size as $S_{l,h}$. For a product of two polynomials of this type with a support of size d, this means that the product can again be computed in time $O(\mathsf{M}(d))$.

3.2. Relaxed multiplication

For the above polynomial multiplication algorithms, we assume that the input polynomials are entirely given from the outset. In specific settings, the input polynomials may be only partially known at some point, and it can be interesting to anticipate the computation of the partial output. This is particularly true when working with (truncated) formal power series $f = f_0 + f_1 z + \cdots \in \mathbb{K}[[z]]$ instead of polynomials, where it is common that the coefficients are given as a stream.

In this so-called "relaxed" (or "online") computation model, the coefficient $(fg)_d$ of a product of two series $f, g \in \mathbb{K}[[z]]$ must be output as soon as $f_0, \ldots, f_d$ and $g_0, \ldots, g_d$ are known. This model has the advantage that subsequent coefficients $f_{d+1}, f_{d+2}, \ldots$ and $g_{d+1}, g_{d+2}, \ldots$ are allowed to depend on the result $(fg)_d$. This often allows us to solve equations involving power series f by rewriting them into *recursive equations* of the form $f = \Psi(f)$, with the property that the coefficient $\Psi(f)_{d+1}$ only depends on earlier coefficients $f_0, \ldots, f_d$ for all d. For instance, in order to invert a power series of the form $1 + zg$ with $g \in \mathbb{K}[[z]]$, we may take $\Psi(f) = 1 - zfg$. Similarly, if $\mathbb{K}$ has characteristic zero, then the exponential of a power series $g \in \mathbb{K}[[z]]$ with $g_0 = 0$ can be computed by taking $\Psi(f) = 1 + \int fg'$.

From a complexity point of view, let $\mathsf{R}(d)$ denote the cost of the relaxed multiplication of two polynomials of degree $<d$. The relaxed model prevents us from directly using fast "zealous" multiplication algorithms from the previous section that are typically based on FFT-multiplication. Fortunately, it was shown in [16, 9] that

$$\mathsf{R}(d) = O(\mathsf{M}(d) \log d). \tag{6}$$

This relaxed multiplication algorithm admits the advantage that it may use any zealous multiplication as a black box. Through the direct use of FFT-based techniques, the following bound has also been established in [20]:

$$\mathsf{R}(d) = d \log d \, \mathrm{e}^{O(\sqrt{\log \log d})}.$$

We make the same classical assumptions ($\mathsf{R}(d)/d$ is increasing and $\mathsf{R}(2d) = O(\mathsf{R}(d))$) as for classical multiplication; this implies in particular that $\mathsf{R}(d) + \mathsf{R}(e) \leqslant \mathsf{R}(d+e)$. It is also natural to assume that $\mathsf{M}(d) \leqslant \mathsf{R}(2d)$, since ordinary multiplications can be done in a relaxed manner.

3.3. Polynomial reduction

Let us now consider a Gröbner basis of an ideal in $\mathbb{K}[X_1, \ldots, X_r]$, or, more generally, an autoreduced tuple $A = (A_0, \ldots, A_n)$ of polynomials in $\mathbb{K}[X_1, \ldots, X_r]$. Then for any $P \in \mathbb{K}[X_1, \ldots, X_r]$, we may compute a relation

$$P = Q_0 A_0 + \cdots + Q_n A_n + R$$

such that R is reduced with respect to A. We call $(Q_0, \ldots, Q_n, R)$ an *extended reduction* of P with respect to A.

The computation of such an extended reduction is a good example of a problem that can be solved efficiently using relaxed multiplication and recursive equations. For a multivariate polynomial T with dense support of any of the types discussed in section 3.1, let $|T|$ denote a bound for the size of its support. With $\mathsf{R}(d)$ as in (6), it has been shown[1] in [17] that the *quotients* $Q_0, \ldots, Q_n$ and the *remainder* R can be computed in time

$$\mathsf{R}(|Q_0 A_0|) + \cdots + \mathsf{R}(|Q_n A_n|) + O(|R|). \tag{7}$$

1. The results from [17] actually apply for more general types of supports, but this will not be needed in this paper.

This implies in particular that the extended reduction can be computed in quasi-linear time in the size of the equation $P = Q_0 A_0 + \cdots + Q_n A_n + R$. However, as pointed out in the introduction, this equation is in general much larger than the input polynomial P.

Extended reductions $(Q_0, \ldots, Q_n, R)$ are far from being unique (only R is unique, and only if A is a Gröbner basis). The algorithm from [17] for the computation of an extended reduction relies on a *selection strategy* that uniquely determines the quotients. More precisely, for every monomial $M \in \mathcal{M}$, we define the set $\mathcal{I}_M := \{i \in \{0, \ldots, n\} : \mathrm{lm}(A_i) \mid M\}$; then we need a rule to chose a particular index $i_M \in \mathcal{I}_M$ (assuming $\mathcal{I}_M$ is non-empty). The initial formulation [17] used the simplest such strategy by taking $i_M = \min \mathcal{I}_M$, but the complexity bound (7) holds for any selection strategy. Now the total size of all quotients $Q_0, \ldots, Q_n$ may be much larger than the size of P for a general selection strategy. As already mentioned in section 2.4, one of the key ingredients of the fast reduction algorithm in this paper is the careful design of a "dichotomic selection strategy" that enables us to control the degrees of the quotients.

Remark 9. The notion of selection strategy is somewhat similar to the concept of *involutive division* introduced for the theory of involutive bases [13], although our definition is more permissive.

4. CONCISE GRÖBNER BASES

The ideal $\langle A, B \rangle$ has a degree $D := n\,m$ and both the reduced Gröbner basis G^{red} and the Gröbner basis G defined by (3–5) take space $\Theta(n\,D)$. The aim of this section is to present an alternative, so-called *concise representation* for G that only takes space $\tilde{O}(D)$ and that can be computed in quasi-linear time $\tilde{O}(D)$ from the generators A and B. We will say that G is a *concise Gröbner basis*, when using this representation. In the next section, we will see that the reduction of a polynomial with respect to a concise Gröbner basis can also be done in quasi-linear time.

4.1. The dichotomic selection strategy

Consider the extended reduction of a polynomial P with respect to our Gröbner basis G:

$$P = Q_0 G_0 + \cdots + Q_n G_n + R.$$

Recall from section 3.3 that the remainder R is unique, but that the quotients $Q_0, \ldots, Q_n$ may depend on the particular "selection strategy" that we use for our reduction.

For each monomial $M \in \mathcal{M}$, the selection strategy should specify the index $i = i_M$ such that M will be reduced against G_i (we set $i_M = -1$ if there is no suitable index). Such a selection strategy naturally extends to terms $\tau = c\,M$ and determines a pair $\Phi_G(\tau) = (i_M, \tau / \mathrm{lt}(G_{i_M}))$ if $i_M \neq -1$ and $\Phi_G(\tau) = (-1, \tau)$ otherwise (here we wrote $\mathrm{lt}(G_{i_M})$ for the leading term of G_{i_M}). Intuitively, if $\Phi_G(\tau) = (i, \tau')$, then the term τ is reduced against G_i and leads to the term τ' in Q_i. The case $i = -1$ corresponds to monomials that are already in normal form with respect to G, which leads to the term τ in the remainder R.

Now for the design of fast reduction algorithms, it is essential to control the degrees of the quotients Q_i. This motivates the introduction of the following *dichotomic selection strategy*. Recall that the 2-*adic valuation* of an integer i is the largest $\lambda \in \mathbb{N}$ such that 2^λ divides i; we will write $\lambda = \mathrm{val}_2(i)$. The idea is to reduce each monomial preferably against one end of the Gröbner basis

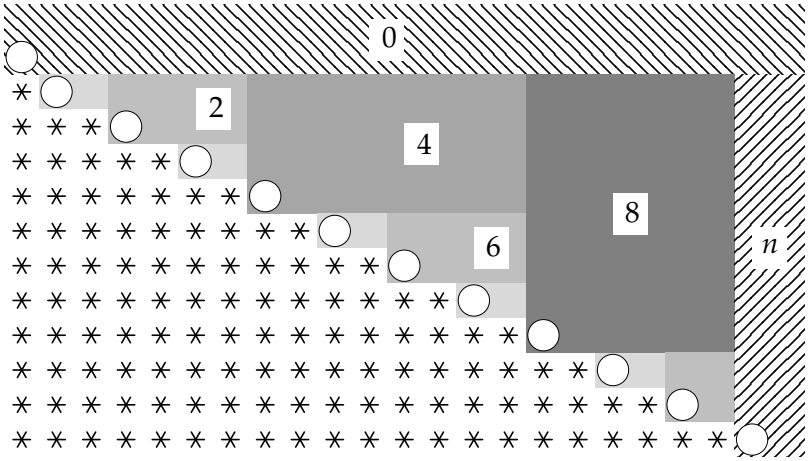

Figure 2. The dichotomic selection strategy for $\deg A = \deg B = 11$. Monomials falling in each area are reduced against the corresponding basis element. Monomials marked by an asterisk (⋇) are already reduced.

(G_0 or G_n), or the G_i such that i has the highest 2-adic valuation. More precisely, the selection strategy is determined by the following function Φ_G on terms:

$$\Phi_G(\tau) := \begin{cases} (0, \tau/\mathrm{lt}(G_0)) & \text{if } \mathrm{lt}(G_0) \text{ divides } \tau \\ (n, \tau/\mathrm{lt}(G_n)) & \text{if } \mathrm{lt}(G_n) \text{ divides } \tau \text{ (and } \mathrm{lt}(G_0) \text{ does not)} \\ (i, \tau/\mathrm{lt}(G_i)) & \text{if } \mathrm{lt}(G_i) \text{ divides } \tau \text{ with } \mathrm{val}_2(i) \text{ maximal} \\ (-1, \tau) & \text{if no } \mathrm{lt}(G_i) \text{ divides } \tau \end{cases}$$

Notice that this definition is non-ambiguous: there is only one index i with $\mathrm{val}_2(i)$ maximal such that $\mathrm{lt}(G_i)$ divides τ. Indeed, if $i < j$ have the same valuation λ, then there is some k with $i < k < j$ and $\mathrm{val}_2(k) > \lambda$. Moreover, if $\mathrm{lt}(G_i)$ and $\mathrm{lt}(G_j)$ both divide τ, then so does $\mathrm{lt}(G_k)$.

This dichotomic selection strategy is illustrated in Figure 2. Again, the white dots (◯) represent the leading term of each basis element. The asterisks (⋇) below the stairs denote monomials that are already in normal form with respect to G (i.e. the canonical basis of $\mathbb{A} := \mathbb{K}[X,Y]/I$). Finally, the areas above the stairs are the sets of terms that will be reduced against each given basis element: the area labelled with the number i contains the monomials that are reduced against G_i. With this selection strategy, most quotients have a very small degree:

LEMMA 10. *Let* $(i, \tau') := \Phi_G(\tau)$. *If* $0 < i < n$, *then* $\deg(\tau') < 3 \times 2^{\mathrm{val}_2(i)} - 1$.

Proof. Let $c X^a Y^b := \tau'$ and $\ell := 2^{\mathrm{val}_2(i)}$. Recall that $\mathrm{lm}(G_0) = Y^n$ and for $0 < i \leqslant n$, we have $\mathrm{lm}(G_i) = X^{m-n-1+2i} Y^{n-i}$. Then we observe that $b < \ell$: otherwise $\mathrm{lt}(G_{i-\ell})$ would divide τ, whereas $\mathrm{val}_2(i-\ell) > \mathrm{val}_2(i)$. A similar reasoning with $G_{i+\ell}$ (or G_n, whenever $i + \ell > n$) shows that $a < 2\ell$. □

4.2. Concise representations of Gröbner bases

In section 2, we defined the basis G using the recurrence relation $G_{k+2} = U_k G_k + V_k G_{k+1}$ for $k = 0, \ldots, n-2$. From this, it is easy to deduce higher order recurrence relations $G_{k+\ell} = U_{k,\ell} G_k + V_{k,\ell} G_{k+1}$ for any k and ℓ with $k + \ell \leqslant n$. It is convenient to write such recurrences in matrix form

$$\begin{pmatrix} G_{k+\ell} \\ G_{k+\ell+1} \end{pmatrix} = M_{k,\ell} \begin{pmatrix} G_k \\ G_{k+1} \end{pmatrix}, \tag{8}$$

where, by convention, $G_{n+1} := 0$ to avoid case distinction. This presentation has the advantage that the $M_{k,\ell}$ can be computed from one another using $M_{k,\ell+t} = M_{k+\ell,t} M_{k,\ell}$. Moreover, the size of the coefficients of the $M_{k,\ell}$ can be controlled as a function of k, ℓ. Notice that similar matrices appear in the half gcd algorithm [12, Chapter 11], which is the fastest known method for the computation of gcds.

DEFINITION 11. *For $k = 2, \ldots, n$, let $u_k Z + v_k := D_{k-2} \operatorname{quo} D_{k-1}$ be the successive quotients in the Euclidean algorithm for the dominant diagonals $D_0 := \operatorname{Diag}(G_0)$ and $D_1 := \operatorname{Diag}(G_1)$ (as in section 2). For each k, ℓ with $k + \ell \leqslant n$, define the matrix $M_{k,\ell}$ by*

$$\begin{aligned}
M_{0,1} &:= \begin{pmatrix} 0 & 1 \\ X^{m-n+1} & -u_2 Y - v_2 X \end{pmatrix}, \\
M_{k,1} &:= \begin{pmatrix} 0 & 1 \\ X^2 & -u_{k+2} Y - v_{k+2} X \end{pmatrix}, \text{ for } 0 < k < n-1, \\
M_{n-1,1} &:= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \\
M_{k,\ell+1} &:= M_{k+\ell,1} M_{k,\ell}.
\end{aligned}$$

PROPOSITION 12. *Let the matrices $M_{k,\ell}$ be as in Definition 11. For all i, k with $i + k \leqslant n$, we then have*

$$\begin{pmatrix} G_{k+\ell} \\ G_{k+\ell+1} \end{pmatrix} = M_{k,\ell} \begin{pmatrix} G_k \\ G_{k+1} \end{pmatrix}.$$

Also, $M_{k,\ell+t} = M_{k+\ell,t} M_{k,\ell}$ for all k, ℓ, t with $k + \ell + t \leqslant n$.

Now consider the polynomials $U_{k,\ell}, V_{k,\ell}, \tilde{U}_{k,\ell}, \tilde{V}_{k,\ell}$ such that

$$M_{k,\ell} = \begin{pmatrix} U_{k,\ell} & V_{k,\ell} \\ \tilde{U}_{k,\ell} & \tilde{V}_{k,\ell} \end{pmatrix}.$$

With the convention that the zero polynomial is homogeneous of any degree, we have

- *For all i, $V_{k,\ell}$ is homogeneous of degree $\ell - 1$ and $\tilde{V}_{k,\ell}$ is homogeneous of degree ℓ.*
- *$U_{0,\ell}$ is homogeneous of degree $m - n - 1 + \ell$ and $\tilde{U}_{0,\ell}$ is homogeneous of degree $m - n + \ell$.*
- *For all $i \geqslant 1$, $U_{k,\ell}$ is homogeneous of degree ℓ and $\tilde{U}_{k,\ell}$ is homogeneous of degree $\ell + 1$.*

Proof. This is immediate, by induction on ℓ, while using the formula $M_{k,\ell+1} = M_{k+\ell,1} M_{k,\ell}$. □

During extended reductions, we have already explained how the dichotomic selection strategy allows us to control the degrees of the quotients. In addition, we will gradually rewrite the linear combination $Q_0 G_0 + \cdots + Q_n G_n$ using our recurrence relations in such a way that we only need a limited number of head terms of each G_i during the evaluation. More precisely, the lower the degree bound for Q_i, the less terms of G_i will be needed. Let us now examine more precisely how many terms of each G_i should be known.

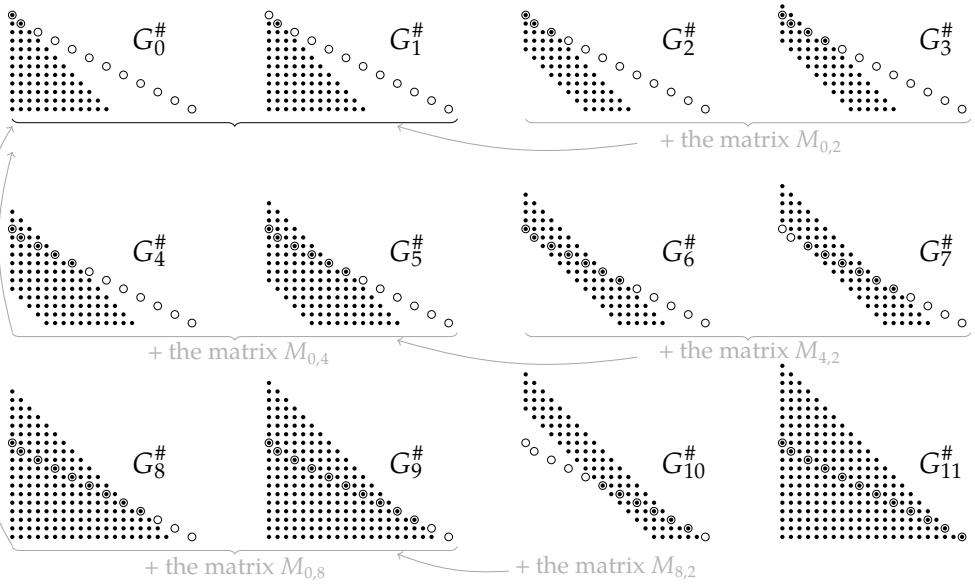

Figure 3. The concise representation of the Gröbner basis when $\deg A = \deg B = 11$.

DEFINITION 13. *Given a polynomial $P \in \mathbb{K}[X,Y]$, we define its **upper truncation with precision p** as the polynomial $\pi_p^\#(P)$ such that*

- $\pi_p^\#(P) \in \langle X,Y\rangle^{\deg P - p}$;
- $\deg(P - \pi_p^\#(P)) < \deg P - p$.

In other words, P and $\pi_p^\#(P)$ have the same terms of degree at least $\deg P - p$, but all terms of $\pi_p^\#(P)$ of degree less than $\deg P - p$ are zero.

With the dichotomic selection strategy, we have $\deg Q_i < 3 \times 2^{\mathrm{val}_2(i)}$ for $0 < i < n$, so it is natural to compute G_i with the same precision. However, the rewriting rules among the G_i involve two consecutive elements, so that G_{2j} and G_{2j+1} need to be known with the same precision. This motivates the following definition:

DEFINITION 14. *The **concise representation** of $G = (G_0, \ldots, G_n)$ consists of the following data:*

- *The sequence of truncated elements $G_0^\#, \ldots, G_n^\#$, where*

$$\begin{aligned} G_i^\# &:= G_i && (i = 0, 1, n) \\ G_i^\# &:= \pi_{p(i)}^\#(G_i), \qquad p(i) := 3 \times 2^{\max(\mathrm{val}_2(i), \mathrm{val}_2(i-1))} && (i = 2, \ldots, n-1) \end{aligned}$$

- *The collection of rewriting matrices*

$$\mathcal{M}_\lambda := (M_{0,2^\lambda}, M_{2^\lambda,2^\lambda}, \ldots, M_{2^\lambda q, r}), \qquad \lambda = 0, \ldots, \lceil \log_2 n \rceil,$$

with $q := (n-1) \operatorname{quo} 2^\lambda$, $r := (n-1) \operatorname{rem} 2^\lambda + 1$, and the matrices $M_{k,\ell}$ as in Definition 11.

An example of a concise representation is given in Figure 3. Notice that the truncated polynomial $G_2^\#$ contains much fewer terms than the corresponding G_2 (as seen in Figure 1). Notice also that the rewriting matrices allow to express some elements in terms of others known with higher precision: for example G_6 and G_7 are expressed in terms of G_4 and G_5, themselves written as a function of G_0 and G_1.

The concise representation requires quasi-linear space with respect to the degree of the ideal:

PROPOSITION 15. *The concise representation requires space* $O(n\,m\log n)$.

Proof. It is easy to see that G_i has degree at most $n+i\leqslant 2n$ in the variable Y, and at most $m+i\leqslant 2m$ in the variable X and in total degree. Then for $i=0,1,n$, each non-truncated element $G_i^{\#}:=G_i$ takes $O(n\,m)$ space. Similarly, for $i=2,\ldots,n-1$, each truncated element $G_i^{\#}$ requires $O(m\,p(i))$ space. For $\lambda=1,\ldots,\lceil\log_2 n\rceil$, there are roughly $2n/2^{\lambda}$ indices i such that $\max(\mathrm{val}_2(i),\mathrm{val}_2(i-1))=\lambda$, so all elements together require $O(m\,n\log n)$ space.

There are $\lceil n/2^{\lambda}\rceil$ elements in $\mathcal{M}_\lambda$, each consisting of four homogeneous polynomials of degree roughly 2^{λ} (by Proposition 12), except for $M_{0,2^\lambda}$ that has two polynomial entries of degree roughly 2^{λ} and two entries of degree roughly $m-n+2^{\lambda}$. Hence $\mathcal{M}_\lambda$ requires $O(m)$ space and the collection of all $\mathcal{M}_\lambda$ takes $O(m\log n)$ space. □

4.3. Computing concise Gröbner bases

The definition of the concise representation as above is constructive, but the order of the computation must be carefully chosen to achieve the desired quasi-linear complexity. First, by exploiting the recurrence relations $M_{k,\ell+t}=M_{k+\ell,t}M_{k,\ell}$, it is easy to compute $\mathcal{M}_{\lambda+1}$ from $\mathcal{M}_\lambda$ using the following auxiliary function:

Algorithm 1
Input: $\mathcal{M}_\lambda$ as in Definition 14.
Output: $\mathcal{M}_{\lambda+1}$ as in Definition 14.

```
Set L := #M_λ.
For each i < L quo 2:
    Set (M_{λ+1})_i := (M_λ)_{2i+1} (M_λ)_{2i}.
If L rem 2 = 1:
    Set (M_{λ+1})_{L quo 2} := (M_λ)_{L−1}.
Return M_{λ+1}.
```

LEMMA 16. *Algorithm 1 is correct and takes time* $O(\mathsf{M}(m))$.

Proof. Recall that $M_{k,\ell+t}=M_{k+\ell,t}M_{k,\ell}$ for all k,ℓ,t with $k+\ell+t\leqslant n$. In particular, taking $\ell=t=2^{\lambda}$ shows that $(\mathcal{M}_{\lambda+1})_i=(\mathcal{M}_\lambda)_{2i+1}(\mathcal{M}_\lambda)_{2i}$ if $2i+1<L-1$. Concerning the last element of $\mathcal{M}_{\lambda+1}$ (that is, $j=L\,\mathrm{quo}\,2-1$ if L is even, $j=L\,\mathrm{quo}\,2$ otherwise), define

$$\begin{aligned} q &:= (n-1)\,\mathrm{quo}\,2^{\lambda},\\ r &:= (n-1)\,\mathrm{rem}\,2^{\lambda}+1,\\ q' &:= (n-1)\,\mathrm{quo}\,2^{\lambda+1},\\ r' &:= (n-1)\,\mathrm{rem}\,2^{\lambda+1}+1. \end{aligned}$$

If $L\,\mathrm{rem}\,2=0$, then q is odd, that is $q'=(q-1)/2$ and $r'=r+2^{\lambda}$, so that $M_{2^{\lambda+1}q',r'}=M_{2^\lambda q,r}M_{2^\lambda(q-1),2^\lambda}$ is indeed the product of the last two elements of $\mathcal{M}_\lambda$. Conversely if $L\,\mathrm{rem}\,2=1$, then $q'=q/2$ and $r'=r$ so that $\mathcal{M}_\lambda$ and $\mathcal{M}_{\lambda+1}$ have the same last element.

The complexity bound is obtained with the same argument as for Proposition 15. □

The algorithm to compute the concise representation can be decomposed into three steps. First a Euclidean algorithm gives recurrence relations of order 1. We next deduce higher order relations using the above algorithm. Finally, one has to compute the truncated basis elements $G_i^{\#}$. Starting with those of highest precision avoids computing unnecessary terms, so that quasi-linear complexity can be achieved.

Algorithm 2
Input: (A,B), two generic bivariate polynomials of total degrees n and m with $n \leqslant m$.
Output: $(G^{\#}, \mathcal{M})$, the concise representation of a Gröbner basis of $I := \langle A,B\rangle$ with respect to $\prec$.

1 Set $G_0^{\#} := A$ and $G_1^{\#} := B \operatorname{rem} A$.
2 Set $D_0 := \mathrm{Diag}(G_0^{\#})$ and $D_1 := \mathrm{Diag}(G_1^{\#})$.
3 For $i = 2,\dots,n$:
4 Set $D_i := D_{i-2} \operatorname{rem} D_{i-1}$ and $u_i Z + v_i := D_{i-2} \operatorname{quo} D_{i-1}$. *// Fail if the quotient has degree >1*
5 If $i=2$, then set $d_i := m-n+1$, otherwise set $d_i := 2$.
6 Set $M_{i-2,1} := \begin{pmatrix} 0 & 1 \\ X^{d_i} & -u_i Y - v_i X \end{pmatrix}$.
7 Set $M_{n-1,1} := \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ and $\mathcal{M}_0 := (M_{0,1},\dots,M_{n-1,1})$.
8 For $\lambda = 0,\dots,\lceil \log_2 n\rceil - 1$:
9 Compute $\mathcal{M}_{\lambda+1}$ from $\mathcal{M}_\lambda$ using Algorithm 1.
10 Compute $\begin{pmatrix} G_n^{\#} \\ 0 \end{pmatrix} := M_{0,n} \begin{pmatrix} G_0^{\#} \\ G_1^{\#} \end{pmatrix}$. *// use $\mathcal{M}_{\lceil \log_2 n\rceil} = (M_{0,n})$*
11 For $\lambda = \lceil \log_2 n\rceil - 1,\dots,1$:
12 For each $j = 2,\dots,n-1$ with $j \operatorname{rem} 2^{\lambda+1} = 2^\lambda$, set *// Use $\mathcal{M}_\lambda = (M_{0,2^\lambda}, M_{2^\lambda,2^\lambda},\dots)$*
13 $$\begin{pmatrix} G_j^{\#} \\ G_{j+1}^{\#} \end{pmatrix} := \pi^{\#}_{3\times 2^\lambda}\left(M_{j-2^\lambda,2^\lambda} \begin{pmatrix} G^{\#}_{j-2^\lambda} \\ G^{\#}_{j-2^\lambda+1} \end{pmatrix}\right).$$
14 Return $G^{\#} := (G_0^{\#},\dots,G_n^{\#})$ and $(\mathcal{M}_0,\dots,\mathcal{M}_{\lceil \log_2 n\rceil})$.

THEOREM 17. *Algorithm 2 is correct and takes time* $O(\mathsf{R}(m^2) + \mathsf{M}(n\,m)\log(n))$

Proof. The reduction $G_1^{\#} := B \operatorname{rem} A$ can be done in a relaxed way in time $O(\mathsf{R}(m^2))$. The first loop (lines 3-6) is clearly correct and each step requires $O(\mathsf{M}(n))$ operations. Alternatively, the successive quotients can all be computed with the fast "half gcd" algorithm (see for example [12, Chapter 11]) using $O(\mathsf{M}(n)\log n)$ operations.

In the last loop on λ (lines 11-13), since the precision decreases at each step and since there is no accidental cancellation, the invariant

$$\begin{pmatrix} G_j^{\#} \\ G_{j+1}^{\#} \end{pmatrix} = \pi^{\#}_{3\times 2^\lambda}\begin{pmatrix} G_j \\ G_{j+1} \end{pmatrix} \text{ for all } j \text{ with } j \operatorname{rem} 2^{\lambda+1} = 2^\lambda$$

holds. Indeed, regarding upper truncations, it is clear that $\pi^{\#}_u(PQ) = \pi^{\#}_u(\pi^{\#}_v(P)\,Q)$ as soon as $u \leqslant v$. Then we have

$$\pi^{\#}_{3\times 2^\lambda}\left(M_{i-2^\lambda,2^\lambda} \begin{pmatrix} G^{\#}_{j-2^\lambda} \\ G^{\#}_{j-2^\lambda+1} \end{pmatrix}\right) = \pi^{\#}_{3\times 2^\lambda}\left(M_{j-2^\lambda,2^\lambda} \begin{pmatrix} G_{j-2^\lambda} \\ G_{j-2^\lambda+1} \end{pmatrix}\right),$$

which proves the correctness. Let us now evaluate the complexity of this loop. For each index j such that $j \operatorname{rem} 2^{\lambda+1} = 2^\lambda$, we have to estimate the support of $G_k^\#$ and $G_{k+1}^\#$ (where $k := j - 2^\lambda$). For $j = 2^\lambda$, $G_k^\#$ and $G_{k+1}^\#$ are completely known, with a support of size $O(mn)$. In all other cases $G_k^\#$ and $G_{k+1}^\#$ are upper truncations, with a support of size $O(m 2^\lambda)$. Consequently, each iteration of the loop requires $O(\mathsf{M}(nm))$ operations. □

5. FAST REDUCTION WITH RESPECT TO CONCISE GRÖBNER BASES

In this section, we show how to use the concise representation to compute an extended reduction with quasi-optimal complexity. Recall that the major obstruction for such a result was that the equation

$$P = Q_0 G_0 + \cdots + Q_n G_n + R. \tag{9}$$

is much larger than the intrinsic complexity of the problem (i.e. the size of A, B, P). On the one hand, the concise representation solves this issue by providing essential information about $G = (G_0, \ldots, G_n)$ using much less space. On the other hand, it is really nontrivial to use it efficiently in a reduction because the fast relaxed algorithm from [17] cannot be used in a blackbox manner. For this reason, the core of the algorithm is completely different to what is found in [18], although some of the techniques are similar.

5.1. Revisiting the relaxed reduction algorithm

Although the fast relaxed algorithm from [17] cannot be used as a blackbox, our algorithm is based on the same principle so it seems important to give an overview of how it works. In fact, our algorithm mimics the behavior of the relaxed reduction algorithm, without the need to expand G. In this section we aim to give the intuition of the classical relaxed reduction, so that it is possible to understand how we exploit the properties of the concise representation later in section 5.3.

Roughly speaking, we will see $Q_0, \ldots, Q_n, R$ as streams of coefficients, and we rewrite equation (9) in an equivalent recursive form

$$(Q_0, \ldots, Q_n, R) = \Psi_{P,G}(Q_0, \ldots, Q_n, R)$$

where the i-th coefficient in each stream depends only of the coefficients 0 to $i-1$ of the streams (and possibly P and G). In this case, it is possible to compute these streams iteratively using relaxed multiplications.

For example, let $f(x), g(x)$ be univariate polynomials with $\deg f > \deg g$, and assume to simplify that g is monic. We wish to find $q(x), r(x)$ such that $f = gq + r$ and $\deg g > \deg r$. This can also we written as

$$x^{\deg g} q + r = f - (g - x^{\deg g})\, q,$$

and we notice that the term of degree k in q is the term of degree $k + \deg g$ in $f - (g - x^{\deg g})\, q$, that depends only on $q_{k+1}, q_{k+2}, \ldots, q_{\deg q}$. This means the equation is indeed recursive if the stream of coefficients starts with the terms of highest degree.

In a more intuitive way, consider the naive schoolbook algorithm: at each step, set

$$q := q + f_{k+\deg g} x^k \quad \text{and} \quad f := f - f_{k+\deg g} x^k g.$$

We can also think of it as "at each step, cancel the leading term in f by adding αx^k to the quotient, then add a compensating $-\alpha\, t(x)$ to f for the next terms", where $t(x) := g(x) - \mathrm{lt}(g)$ is the tail of g. The algorithm from [17] is the multivariate generalization of this idea: we evaluate the polynomial

$$P - Q_0 T_0 - \cdots - Q_n T_n \quad \text{with } T_i := G_i - \mathrm{lt}(G_i)$$

using relaxed multiplications, and this raises a stream of coefficients from which $Q_0, \ldots, Q_n, R$ can be reconstructed. Note that it is necessary to use T_i instead of G_i for the equation to be recursive.

5.2. Exploiting the concise representation

The concise representation contains only truncated variants $G_i^\#$ of the G_i. We can set $T_i^\# := G_i^\# - \mathrm{lt}(G_i^\#)$, then we observe that the formula

$$P - Q_0 T_0^\# - \cdots - Q_n T_n^\#$$

is much smaller than the formula

$$P - Q_0 T_0 - \cdots - Q_n T_n$$

so that relaxed multiplications will compute the former polynomial much faster. However, since the terms of lower degree are dropped, the two streams of coefficients will diverge at some point. To avoid this, we will progressively rewrite equation (9) before this happens. More precisely, as soon as the quotient Q_j is known, the product $Q_j G_j$ is replaced by some $S_k G_k + S_{k+1} G_{k+1}$, where G_k, G_{k+1} are known with precision larger than G_j.

Remark 18. As in [18], we use truncated elements to speed-up the computation, followed by substitutions to maintain the correctness of the result. However, there are major differences in how these ingredients are used.

The concise representation contains the necessary information to perform these substitutions in the form of recurrence relations among the G_k: for any indices k, ℓ with $k + \ell \leqslant n$, we have

$$\begin{pmatrix} G_{k+\ell} \\ G_{k+\ell+1} \end{pmatrix} = M_{k,\ell} \begin{pmatrix} G_k \\ G_{k+1} \end{pmatrix}. \tag{10}$$

For the correctness of the algorithm, we will need the following result:

LEMMA 19. *Let $G_0, \ldots, G_n$ be a Gröbner basis and let the matrices $M_{i,k}$ be as in Definition 11 (for all indices k, ℓ such that $k + \ell \leqslant n$). Given quotients Q_j, Q_{j+1} with $j := k + \ell$, define*

$$(S_k, S_{k+1}) := (Q_j, Q_{j+1})\, M_{k,\ell}.$$

Then we have

$$S_k G_k + S_{k+1} G_{k+1} = Q_j G_j + Q_{j+1} G_{j+1}. \tag{11}$$

Assume now that λ is such that Q_j, Q_{j+1} have degree less than $3\times 2^{\lambda-1}-1$ and $\ell < 2^\lambda$. Then:

- *If $k>0$, then S_k, S_{k+1} have degree less than $3\times 2^\lambda - 1$ and can be computed in time $O(\mathsf{M}(4^\lambda))$.*
- *If $k=0$, then $\deg(S_k) < m-n+3\times 2^\lambda$, $\deg(S_{k+1}) < 3\times 2^\lambda - 1$ and they can be computed in time $O(\mathsf{M}((m-n)\,2^\lambda + 4^\lambda))$.*

Proof. Equation (11) is an immediate consequence of the recurrence relations (10) among the G_k. Similarly, the bounds on $\deg S_k$ are consequences of the degree bounds from Proposition 12. □

5.3. Reduction algorithm

We can now adapt the extended reduction algorithm from [17] to perform these replacements during the computation.

Algorithm 3
Input: $(P, G^\#, \mathcal{M})$, where P is a bivariate polynomial of degree d, and $(G^\#, \mathcal{M})$ is the concise representation of a Gröbner basis G (as in Definition 14).
Output: $(Q_0, \ldots, Q_n, R)$, the extended reduction of P with respect to G.

1 Set $(Q_0, \ldots, Q_n) := (0, \ldots, 0)$
2 Set $(S_0, \ldots, S_n) := (0, \ldots, 0)$ *// new quotients after substitutions as in Lemma 19*
3 Set $P^{\mathrm{subs}} := P$ and $\mathcal{I} := \{i \leqslant n : \deg(G_i) \leqslant d\} \cup \{n\}$ *// $\mathcal{I}$: active indices for relaxed multiplications*
4 For $i = 0, \ldots, n$, set $T_i^\# := G_i^\# - \mathrm{lt}(G_i^\#)$.
5 For $d' = d, \ldots, 0$:
6 $\quad \mathcal{J} := \{i \in \mathcal{I} : (i=0) \vee (i=n) \vee (d' < \deg(G_i) + 3\times 2^{\mathrm{val}_2(i)})\}$ *// keep only the indices with $Q_i \neq 0$*
7 $\quad$ For $a = 0, \ldots, d'$:
8 $\quad\quad$ Let τ be the term of $X^a Y^{d'-a}$ in $P^{\mathrm{subs}} - \sum_{i\in\mathcal{J}} Q_i T_i^\#$, computed in a relaxed manner.
9 $\quad\quad$ Let $(i, \tau') := \Phi_G(\tau)$.
10 $\quad\quad$ If $i < 0$, then update $R \mathrel{+}= \tau'$, else update $Q_i := Q_i + \tau'$.
11 $\quad$ For all j (in decreasing order) such that $d' = \deg(G_j)$: *// See Remark 20*
12 $\quad\quad$ If $j < n$, then update $S_j \mathrel{+}= Q_j$ and $\mathcal{I} := \mathcal{I} \setminus \{j\}$ and $P^{\mathrm{subs}} \mathrel{-}= Q_j G_j^\#$
13 $\quad\quad$ If $1 < j < n$ and $\lambda := \mathrm{val}_2(j) > 0$, then:
14 $\quad\quad\quad$ Set $k := j - 2^\lambda$.
15 $\quad\quad\quad$ Set $(\Delta_k, \Delta_{k+1}) := (S_j, S_{j+1})\, M_{k,2^\lambda}$.
16 $\quad\quad\quad$ Update $(S_k, S_{k+1}) \mathrel{+}= (\Delta_k, \Delta_{k+1})$.
17 $\quad\quad\quad$ Update $P^{\mathrm{subs}} \mathrel{-}= \Delta_k G_k^\# + \Delta_{k+1} G_{k+1}^\# - S_j G_j^\# - S_{j+1} G_{j+1}^\#$
18 $\quad\quad\quad$ Set $(S_j, S_{j+1}) := (0, 0)$.
19 Return $(Q_0, \ldots, Q_n, R)$

Remark 20. Recall that G_0 has degree n and G_i has degree $m+i-1$ for $i \geqslant 1$. Therefore, the loop in lines 11-18 on j such that $d' = \deg(G_j)$ is trivial: the set of such j contains at most 1 element, except when $d' = n = m$, in which case there are 2 elements 0 and 1.

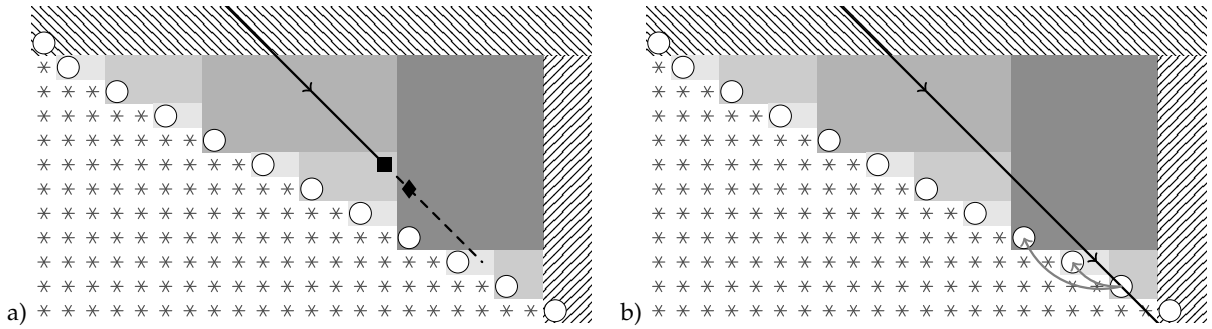

Figure 4. Summary of the reduction algorithm.

The reduction algorithm consists of two main tasks presented in Figure 4. First one has to reduce terms one after the other and update the quotients accordingly (loop in lines 7-10). For example in Figure 4.a), the current term to be reduced is $t := \alpha X^{14} Y^6$, marked by a square; it is reduced against G_6 (with $\mathrm{lt}(G_6) = \beta X^{11} Y^5$) because of the dichotomic selection strategy, then we update the quotient $Q_6 \mathrel{+}= \alpha / \beta X^3 Y$ to cancel this term. During the next step of the loop, the term to be reduced is $\gamma X^{15} Y^5$ marked by a diamond; this time it is reduced against G_8 which implies an update of Q_8. For the second task, one has to rewrite the equation to maintain sufficient precision (loop in lines 11-18); this is possible because a new quotient is now entirely known. In our example on Figure 4.b), we just completed the computation of Q_{10}, so we find S_8, S_9 such that

$$Q_{10} G_{10} = S_8 G_8 + S_9 G_9$$

and we perform the replacement in equation (9); this is represented by the grey arrows in the picture.

Remark 21. Actually, if we just completed the computation of Q_j with j odd, then we wait for the computation of Q_{j-1} to perform both replacements (test in line 13) because it is more practical.

THEOREM 22. *Algorithm 3 is correct and runs in time*

$$O(\mathsf{R}(d^2) + \mathsf{R}(n\,m) \log n + \mathsf{M}(n\,m) \log^2 n).$$

Proof. Let us first explain why the relaxed strategy can indeed be used. We regard the quotients Q_i as streams of coefficients. These coefficients are produced by the updates $Q_i := Q_i + \tau'$ and consumed in the relaxed evaluation of the products $Q_i T_i^{\#}$. Since our reduction process is essentially based on the same recursive functional equation as in [17], the production of coefficients always occurs before their consumption.

We will show that Algorithm 3 computes the same result as the traditional relaxed reduction algorithm from [17], because the term τ that is considered at each step is the same in both cases. More precisely, we must show that for each d', at the start of each iteration of the loop on a (line 7), we have

$$\left(P^{\mathrm{subs}} - \sum_{i \in \mathcal{J}} Q_i T_i^{\#} \right)_{a, d'-a} = \left(P - \sum_{i \leqslant n} Q_i T_i \right)_{a, d'-a} \tag{12}$$

where $T_i := G_i - \mathrm{lt}(G_i)$ is the non-truncated analogous of $T_i^{\#}$. Informally, equation (12) means that Algorithm 3 (left-hand side) and the algorithm from [17] (right-hand side) produce the same streams of coefficients; hence the correctness of [17] implies the correctness of Algorithm 3.

Let us start with the description of a few invariants at the start of the main loop on d':

I1. $P^{\mathrm{subs}} = P - \sum_{i \leqslant n} S_i G_i^{\#}$.

I2. $\sum_{i \leqslant n} S_i G_i = \sum_{i \notin \mathcal{J}} Q_i G_i$.

I3. $\deg(S_i)$ and $\deg(S_{i+1})$ are at most $3 \times 2^{\mathrm{val}_2(i)} - 1$ for all even $i \in \{1, \ldots, n-1\}$.

I4. For some $i_0 \leqslant n+1$, we have $\mathcal{J} = \{0, \ldots, i_0 - 1\} \cup \{n\}$.

I5. With i_0 as above, if $i = n$ or $i > i_0$ or ($i \geqslant i_0$ with i_0 even), then $S_i = 0$.

Invariant I1 is immediate. Invariants I2 and I3 follow from Lemma 19, using $\deg(Q_i) < 3 \times 2^{\mathrm{val}_2(i)} - 1$ by Lemma 10. For invariant I4, we recall that $\deg(G_{i'}) \leqslant \deg(G_{i'+1})$ for all $i' < n$. Finally, invariant I5 is preserved by the loop on lines 11-18: whenever j is removed from $\mathcal{J}$, S_j and S_{j+1} are set to 0 if j is even.

Let us now prove the main claim (12). Notice first that if $0 < i < n$ and $i \in \mathcal{J}$, then $\deg(G_i) \leqslant d'$. Recall also that $\deg(Q_i) < 3 \times 2^{\mathrm{val}_2(i)} - 1$ for $0 < i < n$. This means $\deg(Q_i G_i) < d'$ for $i \in \mathcal{J} \setminus \mathcal{J}$. Since by definition

$$G_0^{\#} = G_0, G_n^{\#} = G_n, \text{and } G_i^{\#} = \pi^{\#}_{3 \times 2^{\mathrm{val}_2(i)}}(G_i) \text{ for } 0 < i < n,$$

we deduce that

$$\left(\sum_{i \in \mathcal{J}} Q_i T_i^{\#} \right)_{a, d'-a} = \left(\sum_{i \in \mathcal{I}} Q_i T_i^{\#} \right)_{a, d'-a} = \left(\sum_{i \in \mathcal{I}} Q_i T_i \right)_{a, d'-a}.$$

To complete the proof (by invariant I1), we show that

$$\left(\sum_{i \leqslant n} S_i G_i^{\#} \right)_{a, d'-a} = \left(\sum_{i \leqslant n} S_i G_i \right)_{a, d'-a} = \left(\sum_{i \notin \mathcal{I}} Q_i G_i \right)_{a, d'-a} = \left(\sum_{i \notin \mathcal{I}} Q_i T_i \right)_{a, d'-a}.$$

For the first identity, we contend that $S_i G_i$ and $S_i G_i^{\#}$ have the same terms of degree d' because of invariants I3 and I5. This is clear if $S_i = 0$, or if $i \leqslant 1$ since $G_0 = G_0^{\#}$ and $G_1 = G_1^{\#}$. Assume therefore that $i > 1$ and $S_i \neq 0$. By invariant I5, the index $i_0 := \min\{i \in \mathbb{N}, i \notin \mathcal{I}\}$ verifies $i \leqslant i_0 < n$, hence i_0 was removed from $\mathcal{I}$ during a previous iteration of the loop on d'. Since $\deg(G_{i'+1}) = \deg(G_{i'}) + 1$ for all $0 < i' < n$, this actually happened during the previous iteration (with $d'+1$ instead of d'). It follows that $\deg(G_i) \leqslant d' + 1 = \deg(G_{i_0})$. By definition of $G_i^{\#}$ and invariant I3, the polynomials $S_i G_i$ and $S_i G_i^{\#}$ have the same terms of degree at least $\deg(G_i) - 3 \times 2^{\mathrm{val}_2(i)} + \deg(S_i)$ if i is even, or at least $\deg(G_i) - 3 \times 2^{\mathrm{val}_2(i-1)} + \deg(S_i)$ if i is odd. In both cases, this degree bound is $\leqslant d'$.

The second identity follows immediately from invariant I2. The last identity follows from the implication $i \notin \mathcal{I} \Rightarrow d' < \deg(G_i)$, so that $Q_i G_i$ and $Q_i T_i$ have the same terms of degree d'.

For the complexity, relaxed multiplications are used to compute the coefficients of the $Q_i T_i^{\#}$, whose support is a subset of the support of $Q_i G_i^{\#}$. Then the relaxed multiplications take time

$$\mathsf{R}(|Q_0 G_0^{\#}|) + \cdots + \mathsf{R}(|Q_n G_n^{\#}|) = O(\mathsf{R}(d^2) + \mathsf{R}(nm) \log n).$$

It remains to evaluate the cost of the zealous multiplications during the rewriting steps. To avoid case distinctions, given an even index $k \in \{0, \ldots, n-1\}$, let $\overline{\mathrm{val}_2}(k)$ be defined as

$$\overline{\mathrm{val}_2}(k) := \begin{cases} \lceil \log_2 n \rceil & \text{if } k = 0 \\ \mathrm{val}_2(k) & \text{otherwise.} \end{cases}$$

Then for every such index k and for every $\lambda < \overline{\mathrm{val}_2}(k)$, there is an update of S_k, S_{k+1}, of cost $O(\mathsf{M}(4^\lambda))$, followed by the evaluation of the products $S_k G_k^\#$ and $S_{k+1} G_{k+1}^\#$, which takes $O(\mathsf{M}(m 2^{\overline{\mathrm{val}_2}(k)}))$ operations. This leads to a total of $O(\mathsf{M}(m 2^{\overline{\mathrm{val}_2}(k)}) \log_2 n)$ operations. Summing over all k, we get a total cost of $O(\mathsf{M}(nm)\log^2 n)$ for all rewriting steps. □

6. APPLICATIONS

Under some regularity assumptions, we provided a quasi-linear algorithm for polynomial reduction, but unlike in [18], it does not rely on expensive precomputations. This leads to significant improvements in the asymptotic complexity for various problems. To illustrate the gain, let us assume to simplify that $n = m$, and neglect logarithmic factors. Then, ideal membership test and modular multiplication are essentially quadratic in n. Also, computing the reduced Gröbner basis has cubic complexity. In all these examples, the bound is intrinsically optimal, and corresponds to a speed-up by a factor n compared to the best previously known algorithms.

6.1. Ideal membership

From any fast algorithms for Gröbner basis computation and (multivariate) polynomial reduction, it is immediate to construct an ideal membership test:

Algorithm 4
Input: (A, B, P), bivariate polynomials of degrees n, m, and d with $n \leqslant m$ and A, B generic.
Output: **true** if $P \in \langle A, B\rangle$, **false** otherwise.

1 Let $(G^\#, \mathcal{M})$ be the concise representation of the Gröbner basis G of $\langle A, B\rangle$ with respect to $\prec$, computed using Algorithm 2.
2 Let $(Q_0, \ldots, Q_n, R)$ be an extended reduction of P modulo G, computed using Algorithm 3.
3 Return **true** if $R = 0$, **false** otherwise.

THEOREM 23. *Algorithm 4 is correct and takes time* $O(\mathsf{R}(m^2 + d^2) + \mathsf{R}(nm)\log n + \mathsf{M}(nm)\log^2 n)$.

6.2. Multiplication in the quotient algebra

We designed a practical representation of the quotient algebra $\mathbb{A} := \mathbb{K}[X, Y]/\langle A, B\rangle$ that does not need more space (up to logarithmic factors) than the algebra itself, while still allowing for efficient computation. The main difference with the terse representation from [18] is that said representation is easy to compute, so that multiplication in $\mathbb{A}$ can be done in quasi-linear time, including the cost for the precomputation:

Algorithm 5
Input: (A, B, P, Q), bivariate polynomials, with A, B generic of degrees $n \leqslant m$ and $P, Q \in \mathbb{A}$ of degree at most $m + n$ (typically in normal form).
Output: $PQ \in \mathbb{A}$ in normal form.

1 Let $(G^\#, \mathcal{M})$ be the concise representation of the Gröbner basis G of $\langle A, B\rangle$ with respect to $\prec$, computed using Algorithm 2.
2 Compute PQ using any (zealous) multiplication algorithm.
3 Let $(Q_0, \ldots, Q_n, R)$ be an extended reduction of PQ modulo G, computed using Algorithm 3.
4 Return R.

THEOREM 24. *Algorithm 5 is correct and takes time* $O(\mathsf{R}(m^2) + \mathsf{R}(mn)\log n + \mathsf{M}(nm)\log^2 n)$.

6.3. Reduced Gröbner basis

Since we can reduce polynomials in quasi-linear time, we deduce a new method to compute the reduced Gröbner basis: first compute the non-reduced basis, together with additional information to allow the efficient reduction (which can be done fast); then reduce each element with respect to the others.

Algorithm 6
Input: (A, B), generic bivariate polynomials of total degrees n and m with $n \leqslant m$.
Output: $G^{\mathrm{red}} := (G_0^{\mathrm{red}}, \ldots, G_n^{\mathrm{red}})$ the reduced Gröbner basis of $\langle A, B\rangle$ with respect to $\prec$.

Let $(G^{\#}, \mathcal{M})$ be the concise representation of G, computed using Algorithm 2.
For all $i = 0, \ldots, n$:
    Set $t_0 := Y^n = \mathrm{lm}(G_0)$ or $t_i := X^{m-n-1+2i} Y^{n-i} = \mathrm{lm}(G_i)$ if $i > 0$.
    Let $(Q_{0i}, \ldots, Q_{ni}, R_i)$ be an extended reduction of t_i modulo G, computed using Algorithm 3.
    Set $G_i^{\mathrm{red}} := t_i - R_i$.
Return $(G_0^{\mathrm{red}}, \ldots, G_n^{\mathrm{red}})$.

THEOREM 25. *Algorithm 6 is correct and takes time* $O(\mathsf{R}(m^2)\, n \log n + n\,\mathsf{M}(nm)\log^2 n)$.

Proof. Clearly G_i^{red} is in the ideal and has the same leading monomial as G_i. Moreover, G_i^{red} is monic and none of its terms is divisible by the leading term of any G_j, $j \neq i$. This proves G^{red} is indeed the reduced Gröbner basis of $\langle A, B\rangle$ with respect to $\prec$. □

7. REFINED COMPLEXITY ANALYSIS

In sections 5 and 6, we purposely simplified our algorithms for convenience of the reader. Although the complexity bounds are satisfactory, there are specific cases in which they are not optimal, and a more subtle analysis is required to improve them. In this section, we detail the missing ingredients and prove the announced complexity bounds from the introduction.

7.1. Optimized algorithm using lazier substitutions

The bound given in Theorem 22 contains an unwanted term $O(\mathsf{M}(nm)\log^2(n))$ that corresponds to the rewriting steps. This contribution is absorbed by the term $O(\mathsf{R}(nm)\log(n))$ when using traditional relaxed multiplication [9, 16] with $\mathsf{R}(d) \asymp \mathsf{M}(d)\log d$, but this is no longer true for faster relaxed algorithms, such as the one from [20]. With some optimizations, it is possible to decrease by a logarithmic factor the cost of the rewriting steps. Then this contribution can be absorbed into the term $O(\mathsf{R}(nm)\log(n))$, independently of the relaxed multiplication algorithm being used.

For the general idea, notice that each time a new quotient Q_j is known, we perform a substitution $(S_k, S_{k+1}) \mathrel{+}= (S_j, S_{j+1})\, M_{k,2^\lambda}$ $(j = k + 2^\lambda)$, followed by a few products of the form $S_k G_k^{\#}$. This means that there are a logarithmic number of products $S_k G_k^{\#}$ for each k. Now up to a few adaptations, it is possible to reduce this to a constant number. Summing over all k, the total cost of rewriting the equation then drops from $O(\mathsf{M}(nm)\log^2 n)$ down to $O(\mathsf{M}(nm)\log n)$.

The modification is essentially as follows:

1. instead of rewriting as soon as Q_j is known, we delay the substitution until Q_k is known;
2. we then perform all substitutions $(S_k, S_{k+1}) \mathrel{+}= (S_j, S_{j+1})\, M_{k,2^\lambda}$ for each $\lambda, j = k + 2^\lambda$;
3. we only perform the multiplications by $G^\#_k, G^\#_{k+1}$ after this loop.

Since the substitution is delayed, it is necessary to increase the precision of $G^\#_j$, to ensure a correct result up to the degree of G_k. If $k > 0$, then we have $\deg(G_j) = \deg(G_k) + 2^\lambda$, so increasing the precision by 2^λ is sufficient. However, if $k = 0$, then $\deg(G_j) = \deg(G_k) + 2^\lambda + m - n$, and $m - n$ may be large; a naive increase of the precision would thus cause an undesirable overhead. A possible workaround is to add a "virtual" basis element $G_{1/2} := X^{m-n} G_0$, that is used only for the substitutions (and we have $\deg(G_j) = \deg(G_{1/2}) + 2^\lambda - 1$ so now the increased precision remains reasonable).

DEFINITION 26. *The **augmented concise representation** has the same content as the concise representation, up to the following:*

- $G^\#_i := \pi^\#_{p(i)}(G_i)$ *for* $i = 2, \ldots, n-1$, *where* $p(i) := 4 \times 2^{\max(\mathrm{val}_2(i), \mathrm{val}_2(i-1))}$ *(instead of* $3 \times 2^{\cdots}$*);*
- *An additional element* $G^\#_{1/2} := X^{m-n} G_0$*;*
- *Additional matrices* $M_{1/2,2^\lambda} := M_{0,2^\lambda} \begin{pmatrix} 1/X^{m-n} & \\ & 1 \end{pmatrix}$ *for each* $\lambda \leqslant \lfloor \log_2 n \rfloor$.

The following theorem is a straightforward adaptation of Proposition 15 and Theorem 17.

THEOREM 27. *The augmented concise representation requires* $O(m n \log n)$ *space and can be computed in time* $O(\mathsf{R}(m^2) + \mathsf{M}(n\,m)\log(n))$ *using a suitable adaptation of Algorithm 2.*

Assuming from now on the augmented concise representation, the loop in lines 11-18 of Algorithm 3 can be modified as follows:

Algorithm 7
Replaces the loop in lines 11-18 of Algorithm 3

```
1   For all k (in decreasing order) such that d' = deg(G_k):          // See Remark 20
2       If k < n, then set S_k := Q_k.
3       If k < n and ⌊k⌋ is even:
4           If k + 1 < n then update J := J \ {k, ⌊k+1⌋} else update J := J \ {k}.
5       If k = 0, then update P^subs -= S_0 G^#_0.
6       If (1 < k < n and Λ := val_2(k) > 0) or k = 1/2, then:
7           If k = 1/2 then set Λ := ⌈log_2 n⌉.
8           For each λ ∈ {1, 2, ..., Λ-1} such that j := ⌊k⌋ + 2^λ < n:
9               Update (S_k, S_⌊k+1⌋) += (S_j, S_{j+1}) M_{k,2^λ}.
10              Update P^subs += S_j G^#_j + S_{j+1} G^#_{j+1}.
11              Set (S_j, S_{j+1}) := (0, 0).
12          Update P^subs -= S_k G^#_k + S_⌊k+1⌋ G^#_⌊k+1⌋.
```

THEOREM 28. *The above modification of Algorithm 3 is correct and runs in time*

$$O(\mathsf{R}(d^2) + \mathsf{R}(n\,m)\log n).$$

Proof. The correctness proof is essentially the same as for Theorem 22, although a bit more technical. Invariant I5 must be modified as follows:

I5*. With i_0 as in Invariant I4, if i even with $i-2^{\mathrm{val}_2(i)} \geqslant i_0$ or $i=n$, then $S_i=0$ and $S_{i+1}=0$.

The rest of the proof is as before except for some degree bounds: if i is even, then the polynomials $S_i G_i$ and $S_i G_i^{\#}$ have the same terms of degree at least $\deg(G_i)-4\times 2^{\mathrm{val}_2(i)}+\deg(S_i)$ (notice the term $4\times 2^{\mathrm{val}_2(i)}$ instead of $3\times 2^{\mathrm{val}_2(i)}$ because the precision in the concise representation was increased). Assuming $S_i\neq 0$, that is $i-2^{\mathrm{val}_2(i)}<i_0$, we have $\deg(G_i)-4\times 2^{\mathrm{val}_2(i)}+\deg(S_i)\leqslant d'$. Similarly if i is odd, then the polynomials $S_i G_i$ and $S_i G_i^{\#}$ have the same terms of degree at least $\deg(G_i)-4\times 2^{\mathrm{val}_2(i-1)}+\deg(S_i)$, which is again $\leqslant d'$.

As to the complexity, it is clear that for each k, there are at most 2 products $S_k G_k^{\#}$: one during the substitution at step k (line 12 in Algorithm 7) and possibly one at step $k-2^{\mathrm{val}_2(k)}$ (line 10). The complexity of the rewriting steps therefore drops to $O(\mathsf{M}(nm)\log n)$, as announced. □

7.2. Improved complexity analysis using refined support bounds

The bound given in Theorem 28 assumes that the input polynomial P has a "triangular" support of degree d; typically the bound is tight if $\mathrm{lm}(P)=Y^d$. However, it may happen that the degree in the variable Y is much smaller than the total degree: this is in particular the case in sections 6.2 and 6.3.

Let us first notice that the degree in the variable Y during the execution of Algorithm 3 can be controlled using the following elementary properties:

LEMMA 29. *We have that for all i, $\deg_Y G_i\leqslant n+i$.*

COROLLARY 30. *If the input polynomial P (in Algorithm 3) verifies $\deg_Y P\leqslant Kn$ for some $K\geqslant 3$, then for each i we have $\deg_Y Q_i G_i\leqslant Kn$.*

Proof. The dichotomic selection strategy imposes that for $i>0$ we have $\deg_Y(Q_i)\leqslant n$, hence $\deg_Y(Q_i G_i)\leqslant 3n$. The result for $i=0$ is obtained from $Q_0 G_0 = P-\sum_{i>0} Q_i G_i$. □

We next extend the discussion from section 3.1 to a new type of supports. Initially, we restricted ourselves to supports of the form $S_{l,h}:=\{M\in\mathcal{M}: l\leqslant \deg M\leqslant h\}$, for which $|S_{l,h}|=\Theta(h\,(h-l))$. We now need to bound the degree in the variable Y independently of the total degree, so we consider suports of the form $S_{l,h,s}:=\{M\in\mathcal{M}: l\leqslant\deg M\leqslant h \text{ and } \deg_Y M\leqslant s\}$. Using the same change of variables as before with $X^a Y^b\to T^{h-a-b}U^b$, it is not hard to check that multiplication can still be done in time $O(\mathsf{M}(|S_{l,h,s}|))$ for such more general supports, and similarly for relaxed multiplication. This allows us to prove the following result:

PROPOSITION 31. *If $K\geqslant 3$ is such that $\deg_Y(P)\leqslant Kn$, then the improved variant of Algorithm 3 (from Theorem 28) runs in time*

$$O(\mathsf{R}(Knd)+\mathsf{R}(nm)\log n).$$

Proof. By Corollary 30, the supports appearing during Algorithm 3 are all of the form $S_{l,h,s}$ with $s\leqslant Kn$, and we have $|S_{l,h,s}|=\Theta(s\,(h-l))$, so that the cost of the relaxed multiplications is

$$\mathsf{R}(|Q_0 G_0^{\#}|)+\cdots+\mathsf{R}(|Q_n G_n^{\#}|)=O(\mathsf{R}(Knd)+\mathsf{R}(nm)\log n).$$

and the cost of the rewriting steps is $O(\mathsf{M}(nm)\log n)=O(\mathsf{R}(nm)\log n)$ as in Theorem 28. □

7.3. Consequences of the refined complexity bounds

We are now in a position to improve the results from section 6. Given a fixed ideal $\langle A, B\rangle$, we may precompute an augmented concise Gröbner basis $(G^{\#}, \mathcal{M})$ once and for all; by Theorem 27, this precomputation takes time $O(\mathsf{R}(m^2) + \mathsf{R}(mn)\log n)$. Theorem 28 then leads to the following improved complexity bound for the ideal membership test from section 6.1.

THEOREM 32. *Given $P \in \mathbb{K}[X,Y]$ with $\deg P \leqslant d$, we may test whether $P \in \langle A, B\rangle$ in time*

$$O(\mathsf{R}(d^2) + \mathsf{R}(mn)\log n),$$

using Algorithm 4, when regarding step 1 as a precomputation.

If $P, Q \in \mathbb{K}[X,Y]$ are in normal form with respect to G, then $\deg(PQ) \leqslant 2(n+m)$ and $\deg_Y(PQ) \leqslant 2n$. Proposition 31 then implies the following quasi-optimal complexity bound for multiplication in the quotient algebra $\mathbb{K}[X,Y]/\langle A, B\rangle$.

THEOREM 33. *Using Algorithm 5, one multiplication in $\mathbb{K}[X,Y]/\langle A, B\rangle$ can be performed in time*

$$O(\mathsf{R}(mn)\log n),$$

when regarding step 1 as a precomputation.

Remark 34. The new bound is quasi-linear in the dimension mn of the quotient algebra. If $n \ll m$, then the new bound improves upon the one from Theorem 24 due to the term $\mathsf{R}(m^2)$. Notice that this term occurred for two reasons. First of all, the computation of the concise Gröbner basis in particular involves the computation of $B \operatorname{rem} A$; we now regard this as a precomputation. The reduction of PQ, which has degree $d = 2(m+n)$, also lead to a cost $O(\mathsf{R}(d^2))$ in Theorem 22. Exploiting the reduced degrees in Y of P and Q, the cost of this reduction is reduced to $O(\mathsf{R}(mn)\log n)$ by Proposition 31.

Similarly, applying the improved bound from Proposition 31 to Algorithm 6, the reduction of $t_0 := Y^n = \operatorname{lm}(G_0)$ or $t_i := X^{m-n-1+2i}Y^{n-i} = \operatorname{lt}(G_i)$ with $i > 0$ takes time $O(\mathsf{R}(nm)\log n)$. This yields:

THEOREM 35. *Using Algorithm 6, the reduced deglex Gröbner basis of $\langle A, B\rangle$ can be computed in time*

$$O(\mathsf{R}(m^2) + \mathsf{R}(nm)\, n\log n).$$

Remark 36. This bound is quasi-linear in $m^2 + n^2 m$, which indeed quasi-optimal if we take into account both the input and output sizes.

8. PERSPECTIVES

A general concern for algorithms with improved asymptotic complexity is the question how well they perform in practice. For the algorithms presented in this work, the second author started a proof-of-concept SAGE [26] implementation that is available at

https://hal.archives-ouvertes.fr/hal-01770408/file/implementation.zip

Concerning the structure of $\mathbb{K}[X,Y]/\langle A,B\rangle$, the computation of concise Gröbner bases using a first implementation of Algorithm 2 outperforms SAGE's built-in Gröbner basis function for degrees $\geqslant 160$. In order to develop implementations of the other algorithms with the announced complexities, it is crucial to build on high quality bivariate polynomial arithmetic. A particularly formidable challenge is to implement fast relaxed arithmetic, which seems hard or impossible to achieve in the high level language from SAGE. So far, the proof-of-concept implementation only allowed us to check the correctness of our other algorithms, by resorting to asymptotically suboptimal lazy (instead of relaxed) arithmetic.

Several theoretical challenges also remain. First we notice that Algorithm 2 fails if the Euclidean algorithm raises a quotient with degree larger than 1. Generically, this should not happen, but this restriction can be limiting in practice, especially in the case of small finite fields. It is then natural to ask how to handle the case of non-normal degree sequences. We conjecture that our algorithm extends (and remains quasi-linear) to the case where the quotients have a bounded degree, with only a logarithmic number of them being larger than 1.

It would also be interesting to extend our ideas to the case of $r > 2$ variables or more general term orderings. Although the dichotomic selection strategy extends in a straightforward manner, we are more pessimistic concerning the other ingredients. Indeed, the bivariate total degree ordering is very special for the reasons mentioned in section 2.

BIBLIOGRAPHY

[1] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. On the complexity of the F5 Gröbner basis algorithm. *Journal of Symbolic Computation*, pages 1–24, sep 2014.

[2] Thomas Becker and Volker Weispfenning. *Gröbner bases: a computational approach to commutative algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993.

[3] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal*. PhD thesis, Universitat Innsbruck, Austria, 1965.

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

[5] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1–3):61–88, 1999.

[6] Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, ISSAC '02, pages 75–83. New York, NY, USA, 2002. ACM.

[7] Jean-Charles Faugère, Pierrick Gaudry, Louise Huot, and Guénaël Renault. Polynomial systems solving by fast linear algebra. *ArXiv preprint arXiv:1304.6039*, 2013.

[8] Jean-Charles Faugère, Patrizia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of zero-dimensional Gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

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

[10] Ralf Fröberg and Joachim Hollman. Hilbert series for ideals generated by generic forms. *Journal of Symbolic Computation*, 17(2):149–157, 1994.

[11] A. Galligo. A propos du théoreme de préparation de weierstrass. In *Fonctions de plusieurs variables complexes*, pages 543–579. Springer, 1974.

[12] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 3rd edition, 2013.

[13] Vladimir P. Gerdt and Yuri A. Blinkov. Involutive bases of polynomial ideals. *Mathematics and Computers in Simulation*, 45(5):519–541, 1998.

[14] D. Harvey and J. van der Hoeven. Faster integer and polynomial multiplication using cyclotomic coefficient rings. Technical Report, ArXiv, 2017. http://arxiv.org/abs/1712.03693.

[15] David Harvey, Joris van der Hoeven, and Grégoire Lecerf. Faster polynomial multiplication over finite fields. Technical Report, ArXiv, 2014. http://arxiv.org/abs/1407.3361.

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

[17] J. van der Hoeven. On the complexity of polynomial reduction. In I. Kotsireas and E. Martínez-Moro, editors, *Proc. Applications of Computer Algebra 2015*, volume 198 of *Springer Proceedings in Mathematics and Statistics*, pages 447–458. Cham, 2015. Springer.

[18] J. van der Hoeven and R. Larrieu. Fast reduction of bivariate polynomials with respect to sufficiently regular Gröbner bases. Technical Report, HAL, 2018. http://hal.archives-ouvertes.fr/hal-01702547.

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

[20] Joris van der Hoeven. Faster relaxed multiplication. In *Proc. ISSAC '14*, pages 405–412. Kobe, Japan, Jul 2014.

[21] Romain Lebreton, Eric Schost, and Esmaeil Mehrabi. On the complexity of solving bivariate systems: the case of non-singular solutions. In *ISSAC: International Symposium on Symbolic and Algebraic Computation*, pages 251–258. Boston, United States, Jun 2013.

[22] Ernst Mayr. Membership in polynomial ideals over $\mathbb{Q}$ is exponential space complete. *STACS 89*, pages 400–406, 1989.

[23] Guillermo Moreno-Socías. Degrevlex Gröbner bases of generic complete intersections. *Journal of Pure and Applied Algebra*, 180(3):263–283, 2003.

[24] A. Schönhage. Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Infor.*, 7:395–398, 1977.

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

[26] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.0)*. 2017. https://www.sagemath.org.