A new efficient algorithm for computing Gröbner bases (F_4)

Jean-Charles Faugère[1]
LIP6/CNRS Université Paris VI
case 168, 4 pl. Jussieu, F-75252 Paris Cedex 05
E-mail: jcf@posso.lip6.fr
January 20, 1999

Abstract

This paper introduces a new efficient algorithm for computing Gröbner bases. To avoid as much as possible intermediate computation, the algorithm computes successive truncated Gröbner bases and it replaces the classical polynomial reduction found in the Buchberger algorithm by the simultaneous reduction of several polynomials. This powerful reduction mechanism is achieved by means of a symbolic precomputation and by extensive use of sparse linear algebra methods. Current techniques in linear algebra used in Computer Algebra are reviewed together with other methods coming from the numerical field. Some previously untractable problems (Cyclic 9) are presented as well as an empirical comparison of a first implementation of this algorithm with other well known programs. This comparison pays careful attention to methodology issues. All the benchmarks and CPU times used in this paper are frequently updated and available on a Web page. Even though the new algorithm does not improve the worst case complexity it is several times faster than previous implementations both for integers and modulo p computations.

1 Introduction

In view of the progress already achieved and the promising potential of current and planned algorithms, polynomial solving could become one of the more attractive application of Computer Algebra: practical problems can be solved, the algorithms are competitive with numerical methods. The main conclusion to be drawn from practice and experience of solving polynomial systems coming from various fields (industrial [Fau98b] problems, pure mathematics [Fro96]) is the following: first of all, even though the computation of a Gröbner basis is a crucial point it must be emphasized that it is only *one* step in the full solving process (change of ordering, triangular systems, real or numerical roots are complementary tools); secondly classical Buchberger algorithms [Buc65, Buc70, Buc85] must be improved since even the best implementations often do not succeed to compute Gröbner bases from big problems. This paper is concerned with describing a new algorithm (whose name is F_4) for computing Gröbner basis. Even if the algorithm works for any admissible ordering, the algorithm F_4 has been designed to be efficient for a degree reverse lexicographical ordering (DRL); computing efficiently a lexicographical Gröbner basis from an already computed Gröbner basis being the task of another algorithm. Paradoxically, if the Buchberger algorithm without optimizations is very simple to describe it becomes much harder to understand how to improve a real implementation. By and large, however, it may

[1]Work partially supported by EEC project FRISCO LTR 21.024.

eventually be possible to suggest two improvements: since 90% of the times is spent computing zero it would be useful to have more powerful criterion to remove useless critical pairs [Buc79] (a powerful theoretical criteria exists but it is too costly); this crucial aspect of the problem is **not studied in this paper**, but is implemented in another algorithm (F_5 [Fau98a]). The second improvement is concerned with strategies: during a Gröbner computation, several choices can be made (select a critical pair, choose a reductor) and even if strategies have been proposed ([Gio91] or even [Ger95]) the heuristics which they rely on could not be satisfactorily explained. So it is difficult to be convinced that they are optimal optimizations. Another bad consequence is that it is very difficult to (massively) parallelize the Buchberger algorithm because the sugar (for instance) strategy imposes a strong sequential ordering. The primary objective of this paper is to propose a more powerful reduction algorithm. For that purpose we will reduce *simultaneously* several polynomials by a list of polynomials by using linear algebra techniques which ensure a global view of the process.

The plan of the paper is as follows. The main Section 2 is devoted to presenting the new algorithm. This section has been divided into several parts: first (2.1), we review the necessary mathematical notations (we make the choice to use the same notations as in the book [Bec93]) and in 2.2 we establish the link between linear algebra (matrices) and polynomial algebra. Then we present (2.3) a basic version of the algorithm without any criteria to eliminate useless pairs. A improved version of the algorithm including the Buchberger criteria is then given in 2.4. We close this section in 2.5 by motivating the choice of a good selection strategy (it seems that selecting all critical pairs with a minimal exponent is a good strategy). Since the algorithm relies heavily on linear algebra, Section 3 contains a short survey of linear algebra techniques we have implemented. A first version of this algorithm has been implemented in C in a new small system called FGb (for Fast Gb). In Section 4 we report an experimental evaluation of this first implementation. The best Gröbner bases programs are compared on a set of well known benchmarks and industrial problems. Finally, in Section 5 we outline the main features of the algorithm along with a list of possible related works and open issues.

The name of this algorithm is simply algorithm number 4. In the rest of this paper F_4 stands for this algorithm.

2 Description of the F_4 algorithm

2.1 Standard notations

We use the notations of [Bec93] for basic definitions: R is the ground ring, $R[\underline{x}] = R[x_1, \ldots, x_n]$ is the polynomial ring. We denote by $T(x_1, \ldots, x_n)$, or simply by T, the set of all terms in these variables. We choose $<$ an admissible ordering on T. If $t = x_1^{\alpha_1} \cdots x_n^{\alpha_n} \in T$, then the *total degree* of t is defined as $\deg(t) = \sum_{i=1}^{n} \alpha_i$. Now let $f \in R[\underline{x}]$, $f \neq 0$, so that $f = \sum c(\alpha_1, \ldots, \alpha_n) x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ (where $c(\alpha_1, \ldots, \alpha_n)$ are elements of R). Then we define the set $M(f)$ of *monomials of* f as $M(f) = \{c(\alpha_1, \ldots, \alpha_n) x_1^{\alpha_1} \cdots x_n^{\alpha_n} \mid c(\alpha_1, \ldots, \alpha_n) \neq 0\}$. The set $T(f)$ of *terms of* f is $T(f) = \{x_1^{\alpha_1} \cdots x_n^{\alpha_n} \mid c(\alpha_1, \ldots, \alpha_n) \neq 0\}$. The *total degree* of $f \neq 0$ is defined as $\deg(f) = max\,\{\deg(t) \mid t \in T(f)\}$. We define the *head term* $\mathrm{HT}(f)$, the *head monomial* $\mathrm{HM}(f)$, and the *head coefficient* $\mathrm{HC}(f)$ of f w.r.t. $<$ as follows: $\mathrm{HT}(f) = max(T(f))$, $\mathrm{HM}(f) = max(M(f))$, and $\mathrm{HC}(f) =$ the coefficient of $\mathrm{HM}(f)$. If F is a subset of $R[\underline{x}]$ we can extend the definition $\mathrm{HT}(F) = \{\mathrm{HT}(f) \mid f \in F\}$, $\mathrm{HM}(F) = \{\mathrm{HM}(f) \mid f \in F\}$ and $T(F) = \{T(f) \mid f \in F\}$; $Id(F)$ denotes the ideal generated by F.

Let $f, g, p \in R[\underline{x}]$ with $p \neq 0$, and let F be a finite subset of $R[\underline{x}]$. Then we say that

- *f reduces to g modulo p* (notation $f \underset{p}{\longrightarrow} g$), if $\exists t \in T(f), \exists s \in T$ such that $s * HT(p) = t$ and $g = f - \frac{a}{HC(p)} * s * p$ where a is the coefficient of t in p.
- *f reduces to g modulo P* (notation $f \underset{P}{\longrightarrow} g$), if $f \underset{p}{\longrightarrow} g$ for some $p \in P$.
- *f is reducible modulo p* if there exists $g \in R[\underline{x}]$ such that $f \underset{p}{\longrightarrow} g$.
- *f is reducible modulo P* if there exists $g \in R[\underline{x}]$ such that $f \underset{P}{\longrightarrow} g$.
- *f is top reducible modulo P* if there exists $g \in R[\underline{x}]$ such that $f \underset{P}{\longrightarrow} g$ and $\mathrm{HT}(g) < \mathrm{HT}(f)$.
- $f \overset{*}{\underset{P}{\longrightarrow}} g$ is the reflexive-transitive closure of $\underset{P}{\longrightarrow}$.
- The *S-polynomial* of f and g is defined as

$$spol(f, g) = HC(g)\frac{lcm(HT(f), HT(g))}{HT(f)}f - HC(f)\frac{lcm(HT(f), HT(g))}{HT(g)}g$$

2.2 Linear algebra and polynomials.

Definition 2.1 *By convention if M is a $s \times m$ matrix, $M_{i,j}$ is j th element of the i th row of M. If $T_M = [t_1, \ldots, t_m]$ an ordered set of terms, let $(\epsilon_i)_{i=1,\ldots,m}$ be the canonical basis of R^m, we consider the linear map $\varphi_{T_M} : V_{T_M} \longrightarrow R^m$ (where V_{T_M} is the submodule of $R[\underline{x}]$ generated by T_M) such that $\varphi_{T_M}(t_i) = \epsilon_i$. The reciprocal function will be denoted by ψ_{T_M}. The application ψ_{T_M} allows to interpret vectos of R^n as polynomials. We note by (M, T_M) a matrix with such an interpretation.*

Definition 2.2 *If (M, T_M) is a $s \times m$ matrix with such an interpretation, then we can construct the set of polynomials:*

$$Rows(M, T_M) := \{\psi_{T_M}(row(M, i)) \mid i = 1, \ldots, s\} \setminus \{0\}$$

where $row(M, i)$ is the i-th row of M (an element of R^m). Conversely, if l is a list of polynomials and T_l an ordered set of terms we can construct an $s \times m$ matrix A (where $s = \mathrm{size}(l)$, $m = \mathrm{size}(T_l)$):

$$A_{i,j} := \mathrm{coeff}(l[i], T_l[j]) \; i = 1, \ldots, s \; j = 1, \ldots, m$$

We note $A^{(l,T_l)}$ the matrix $(A_{i,j})$.

$$
M = \begin{array}{c} \\ f_{i_1} \\ f_{i_2} \\ \vdots \\ f_{i_{2k}} \\ f_{i_{2k+1}} \\ f_{i_s} \end{array}
\begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & \cdots \end{array} \\
\begin{pmatrix}
\times & \times & \times & \cdots & \\
\times & \times & \times & \cdots & \\
\vdots & \vdots & \vdots & \cdots & \\
\times & \times & \times & \cdots & \\
\times & \times & \times & \cdots & \\
\times & \times & \times & \cdots &
\end{pmatrix} \end{array} \qquad (1)
$$

Definition 2.3 *Let M be a $s \times m$ matrix, and $\underline{Y} = [Y_1, \ldots, Y_m]$ new variables. Then $F = Rows(M, \underline{Y})$ is a set of equations, so we can compute $\tilde{F}$ a reduced Gröbner basis of F for a lexicographical ordering such that $Y_1 > \ldots > Y_m$. From this basis we can reconstruct a matrix $\tilde{M} = A^{(\tilde{F}, \underline{Y})}$. We called $\tilde{M}$ the (unique) row echelon form*[1] *of M. We say also that $\tilde{F}$ is a row echelon basis of F.*

$$
\tilde{M} = \begin{array}{c} \\ f_{j_1} \\ f_{j_2} \\ \\ f_{j_l} \\ f_{j_{l+1}} \\ \\ f_{j_m} \end{array}
\begin{array}{c} \begin{array}{cccccccc} t_1 & t_2 & & t_k & t_{k+1} & & t_m \end{array} \\
\begin{pmatrix}
1 & 0 & \ldots & 0 & \times & \cdots & \times \\
0 & 1 & \ldots & 0 & \times & \cdots & \times \\
 & & \ddots & & \times & \cdots & \times \\
0 & 0 & \cdots & 1 & \times & \cdots & \times \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
 & & \vdots & & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0
\end{pmatrix} \end{array} \qquad (2)
$$

where $\times$ denotes a possibly non zero element.

In the case of polynomials we have a similar definition:

Definition 2.4 *Let F be a finite subset of $R[\underline{x}]$ and $<$ an admissible ordering. We define $T_<(F)$ to be $Sort(\{T(f) \mid f \in F\}, <)$, $A := A^{(F, T_<(F))}$ and $\tilde{A}$ the row echelon form of A. We say that $\tilde{F} = Rows(\tilde{A}, T_<(F))$ is the row echelon form of F w.r.t. $<$.*

Elementary properties of row echelon matrices are summarized by the following theorem:

Theorem 2.1 *Let M be a $s \times m$ matrix, and $\underline{Y} = [Y_1, \ldots, Y_m]$ new variables, $F = Rows(M, \underline{Y})$, $\tilde{M}$ the row echelon form of M, $\tilde{F} = Rows(\tilde{M}, \underline{Y})$. We define*

$$
\begin{aligned}
\tilde{F^+} &= \left\{ g \in \tilde{F} \mid \mathrm{HT}(g) \notin \mathrm{HT}(F) \right\} \\
\tilde{F^-} &= \tilde{F} \backslash \tilde{F^+}
\end{aligned}
$$

[1] In some computer algebra system, this is the only way to compute a row echelon form !

For any subset F_- of F such that $size(F_-) = size(\mathrm{HT}(F))$ and $\mathrm{HT}(F_-) = \mathrm{HT}(F)$, then $G = \tilde{F^+} \cup F_-$ is a triangular basis of the $R-$module V_M generated by F. That is to say, for all $f \in V_M$ there exist $(\lambda_k)_k$ elements of R and $(g_k)_k$ elements of G such that $f = \sum_k \lambda_k g_k$, $\mathrm{HT}(g_1) = \mathrm{HT}(f)$ and $\mathrm{HT}(g_k) > \mathrm{HT}(g_{k+1})$.

Proof Since the head terms of G are pairwise distinct, G is linearly independent. We claim that it is also a generating system of V_M. Suppose for a contradiction that there exists $f \in V_M$ such that $f \xrightarrow[G]{*} f' \neq 0$. By definition of a Gröbner basis, $f' \xrightarrow[\tilde{F}]{*} 0$, consequently f' is top reducible modulo $\mathrm{HT}(\tilde{F}) = \mathrm{HT}(\tilde{F^+}) \cup \mathrm{HT}(\tilde{F^-}) = \mathrm{HT}(\tilde{F^+}) \cup \mathrm{HT}(F_-) = \mathrm{HT}(G)$, so that f' is top-reducible modulo G. This is a contradiction. □

We can transpose immediately the theorem for polynomials:

Corollary 2.1 *Let F be a finite subset of E and $<$ an admissible ordering, and $\tilde{F}$ the row echelon form of F w.r.t. $<$. We define*

$$\tilde{F^+} = \left\{ g \in \tilde{F} \mid \mathrm{HT}(g) \notin \mathrm{HT}(F) \right\}$$

For all subset F_- of F such that size(F_-)=size($\mathrm{HT}(F)$) and $\mathrm{HT}(F_-) = \mathrm{HT}(F)$, then $G = \tilde{F^+} \cup F_-$ is a triangular basis of V_M the $R-$module generated by F. For all $f \in V_M$ there exist $(\lambda_k)_k$ elements of R and $(g_k)_k$ elements of G such that $f = \sum_k \lambda_k g_k$, $\mathrm{HT}(g_1) = \mathrm{HT}(f)$ and $\mathrm{HT}(g_k) > \mathrm{HT}(g_{k+1})$.

2.3 The F_4 algorithm

It is well known that during the execution of the Buchberger algorithm, one has a lot of choices:

- select a critical pair in the list of critical pairs.
- choose one reductor among a list of reductors when reducing a polynomial by a list of polynomials.

Buchberger[Buc65] proves that these choices are not important for the correctness of the algorithm, but it is well known that these choices are crucial for the total time computation. Moreover the best strategies [Gio91] inspect only the leading terms of the polynomials to make a choice. Consider the case where all the input polynomials have the same leading term. In that case, all the critical pairs are equal and it is not possible to take a decision. In some sense this problem can be corrected in a simple and surprising way: *we make no choice.* More precisely instead of choosing *one* critical pair at each step, we select a *subset* of critical pairs at the same time. So, in fact, we are delaying the necessary choices in a second step of the algorithm, the linear algebra part of the algorithm

Definition 2.5 *A critical pair of two polynomials (f_i, f_j) is an element of $T^2 \times R[\underline{x}] \times T \times R[\underline{x}]$, $Pair(f_i, f_j) := (lcm_{ij}, t_i, f_i, t_j, f_j)$ such that*

$$lcm(Pair(f_i, f_j)) = lcm_{ij} = \mathrm{HT}(t_i f_i) = \mathrm{HT}(t_j f_j) = lcm(\mathrm{HT}(f_i), \mathrm{HT}(f_j))$$

Definition 2.6 *We say that the degree of the critical pair* $p_{i,j} = Pair(f_i, f_j)$, $deg(p_{i,j})$, *is* $deg(lcm_{i,j})$. *We define the two projections* $Left(p_{i,j}) := (t_i, f_i)$ *and* $Right(p_{i,j}) := (t_j, f_j)$. *If* $(t, p) \in T \times R[\underline{x}]$ *then we note* $\mathrm{mult}((t, p))$ *the evaluated product* $t * p$.

We have now the tools needed to present the basic version of our algorithm. All the matrices occuring in following algorithms are the representation of a list of polynomials through the set of all their terms, as explained in Definition 2.2.

Algorithm $F4$

Input: $\begin{cases} F \text{ a finite subset of } R[\underline{x}] \\ \mathcal{S}el \text{ a function } List(Pairs) \to List(Pairs) \text{ such that } \mathcal{S}el(l) \neq \emptyset \text{ if } l \neq \emptyset \end{cases}$

Output: a finite subset of $R[\underline{x}]$.

$G := F$, $\tilde{F}_0^+ := F$ and $d := 0$

$P := \{Pair(f, g) \mid f, g \in G \; with \; f \neq g\}$

while $P \neq \emptyset$ **do**

  $d := d + 1$

  $P_d := \mathcal{S}el(P)$

  $P := P \backslash P_d$

  $L_d := Left(P_d) \cup Right(P_d)$

  $\tilde{F}_d^+ := Reduction(L_d, G)$

  for $h \in \tilde{F}_d^+$ **do**

    $P := P \cup \{Pair(h, g) \mid g \in G\}$

    $G := G \cup \{h\}$

return G

We have to extend the reduction of a polynomial modulo a subset of $R[\underline{x}]$, to the reduction of a subset of $R[\underline{x}]$ modulo another subset of $R[\underline{x}]$:

Reduction

Input: $\begin{cases} L \text{ a finite subset of } T \times R[\underline{x}] \\ G \text{ a finite subset of } R[\underline{x}] \end{cases}$

Output: a finite subset of $R[\underline{x}]$ (possibly an empty set).

  $F :=$ Symbolic Preprocessing(L, G)

  $\tilde{F} :=$ Reduction to Row Echelon Form of F w.r.t. $<$

  $\tilde{F}^+ := \left\{ f \in \tilde{F} \mid HT(f) \notin HT(F) \right\}$

return $\tilde{F}^+$

Remark 2.1 *By Lemma 2.1, we will see that an equivalent (but slower) definition of* $\tilde{F}^+$ *could be* $\tilde{F}^+ := \left\{ f \in \tilde{F} \mid f \text{ top irreducible by } G \right\}$.

We have now to describe the main function of our algorithm, that is to say the consruction of the "matrix" F. This subalgorithm can be viewed as an usual reduction of all the considered polynomials if we replace the standard arithmetic by: let $0 \neq x, 0 \neq y \in R$, then $x + y = 1$, $x * y = 1$, $x * 0 = 0$ and $x + 0 = 1$. So this is really a *symbolic* preprocessing.

Symbolic Preprocessing

Input: $\begin{cases} L \text{ a finite subset of } T \times R[\underline{x}] \\ G \text{ a finite subset of } R[\underline{x}] \end{cases}$

Output: a finite subset of $R[\underline{x}]$.

$F := \{t * f \mid (t, f) \in L\}$

$Done := \mathrm{HT}(F)$

while $\mathrm{T}(F) \neq Done$ **do**

  m an element of $\mathrm{T}(F) \backslash Done$

  $Done := Done \cup \{m\}$

  if m top reducible modulo G **then**

    $m = m' * \mathrm{HT}(f)$ for some $f \in G$ and some $m' \in T$

    $F := F \cup \{m' * f\}$

return F

Remark 2.2 *It seems that the initial values of Done should be $\emptyset$ but in all application of this function the result is in fact the same with less iterations.*

Remark 2.3 *The symbolic preprocessing is very efficient since its complexity is linear in the size of its outout if $size(G)$ is smaller than the final size of tf $T(F)$ which is usually the case.*

Lemma 2.1 *Let G be finite subset of $R[\underline{x}]$, L be the image by* mult *of a finite subset of $T \times G$ and $\tilde{F}^+ = Reduction(L, G)$. Then for all $h \in \tilde{F}^+$, $\mathrm{HT}(h) \notin Id(\mathrm{HT}(G))$.*

Proof Let F the set computed by the algorithm Symbolic Preprocessing(L, G). Assume for a contradiction that $\exists h \in \tilde{F}^+$ such that $t = \mathrm{HT}(h) \in Id(\mathrm{HT}(G))$. Hence $\mathrm{HT}(g)$ divides t for some $g \in G$. So t is in $T(\tilde{F}^+) \subset T(\tilde{F}) \subset T(F)$ and is top reducible by g, hence $\frac{t}{\mathrm{HT}(g)} * g$ is inserted in F by Symbolic Preprocessing (or another product with the same head term). This contradicts the fact that $HT(h) \notin HT(F)$. $\square$

The following lemma is useful to proof the correctness of the algorithm.

Lemma 2.2 *Let G be finite subset of $R[\underline{x}]$, L be the image by* mult *of a finite subset of $T \times G$ and $\tilde{F}^+ = Reduction(L, G)$. Then $\tilde{F}^+$ is a subset of $Id(G)$. Moreover for all f in the $R-$module generated by L, $f \xrightarrow[G \cup \tilde{F}^+]{*} 0$*

Proof Apply the Corollary 2.1 to F the set generated by Symbolic Preprocessing(L, G). Clearly F is a subset of $F \cup Id(G)$, but it is obvious that L is a subset of $Id(G)$, so that F is a subset of $Id(G)$. Hence any F_- fulfilling the hypothesis of Theorem 2.1 is a subset of $Id(G)$. This conclude the proof of the lemma since the $R-$module generated by L is a submodule of the $R-$module generated by F. $\square$

Remark 2.4 *Let G be a finite subset of $R[\underline{x}]$. It is possible that $f \xrightarrow[G]{*} 0$ but that $NormalForm$ $(f, G) \neq 0$ where $NormalForm$ is the reduction which is used in Buchberger algorithm. The reason for that is that the result of $NormalForm$ depends on many choices (strategies).*

Theorem 2.2 *The algorithm F_4 computes a Gröbner basis G in $R[\underline{x}]$ such that $F \subseteq G$ and $Id(G) = Id(F)$.*

Proof *Termination:* Assume for a contradiction that the **while**-loop does not terminate. We see that there exists an ascending sequence (d_i) of natural numbers such that $\tilde{F}_{d_i}^+ \neq \emptyset$ for all i. Let say that $q_i \in \tilde{F}_{d_i}^+$ (hence q_i can be any element in $\tilde{F}_{d_i}^+$). Let U_i be $U_{i-1} + Id(\mathrm{HT}(q_i))$ for $i > 1$ and $U_0 = (0)$. By Lemma 2.1 we have $U_{i-1} \subsetneq U_i$. This contradicts the fact that $R[\underline{x}]$ is noetherian.

Correctness: We have $G = \cup_{d \geq 0} \tilde{F}_d^+$. We claim that the following are loop invariants of the **while**-loop: G is a finite subset of $R[\underline{x}]$ such that $F \subset G \subset Id(F)$, and $spol(g_1, g_2) \xrightarrow[G]{*} 0$ for all $g_1, g_2 \in G$ such that $\{g_1, g_2\} \notin P$. The first claim is an immediate consequence of the first part of Lemma 2.2. For the second one, if $\{g_1, g_2\} \notin P$, this means that $Pair(g_1, g_2) = (lcm_{1,2}, t_1, g_1, t_2, g_2)$ has been selected in a previous step (say d) by the function $\mathcal{S}el$. Hence $t_1 * g_1$ and $t_2 * g_2$ are in L_d, so $spol(g_1, g_2)$ is an element of the R-module generated by L_d hence by Lemma 2.2 $spol(g_1, g_2) \xrightarrow[G]{*} 0$. □

Remark 2.5 *If $size(\mathcal{S}el(l)) = 1$ for all $l \neq \emptyset$ then the algorithm F_4 is the Buchberger algorithm. In that case the $\mathcal{S}el$ function correspond to the selection strategy in the Buchberger algorithm.*

Example 2.1 *One might wonder why in the proof of the termination of the algorithm we consider only one element of $\tilde{F}_d^+$ and not the whole $\tilde{F}_d^+$.*

If $x > y > z$ for a lexicographical ordering, $F = [f_1 = xy^2 + 1, f_2 = xz^2 + 1, f_3 = y^3 + y^2]$ and $\mathcal{S}el$ = identity, we find $P_1 = \{Pair(f_1, f_2), Pair(f_2, f_3), Pair(f_1, f_3)\}$ and $\tilde{F}_1^+ = \{y^2 - z^2, y + 1\}$ so that $Id(\mathrm{HT}(\tilde{F}_1^+)) = \{y\}$. So contrarily to Buchberger Algorithm this not true that after each operation $G' := G \cup \{h\}$, we have $Id(\mathrm{HT}(G')) \supsetneq Id(\mathrm{HT}(G))$.

2.4 Buchberger criteria. Improved F_4 algorithms

In order to obtain an efficient algorithm we need to insert into the previous algorithm the Buchberger Criteria. Since it is not the subject of this paper to improve the Buchberger Criteria we will use a standard implementation of these criteria such as the Gebauer and Moller installation [GM88]:

Buchberger Criteria

Specification: $\begin{cases} (G_{new}, P_{new}) := Update(G_{old}, P_{old}, h) \\ \text{Update of critical pair list and ideal basis (see [Bec93] p.230)} \end{cases}$

Input: $\begin{cases} \text{a finite subset } G_{old} \text{ of } R[\underline{x}] \\ \text{a finite set } \mathrm{P}_{\mathrm{old}} \text{ of critical pairs of } R[\underline{x}] \\ 0 \neq h \in R[\underline{x}] \end{cases}$

Output: a finite subset of $R[\underline{x}]$ and a list of critical pairs.

In the previous version of the algorithm we used only *some* rows of the reduced matrix (the sets $\tilde{F}^+$), rejecting the rows which were already in the original matrix(the sets TF). In the new version of the algorithm we keep these useless rows, and we try to replace some products $m * f$ occuring in the rows of the "matrix" F by a new "equivalent" product $m' * f'$ with $m \geq m'$. This is the task of the function $\mathrm{Simplify} : T \times R[\underline{x}] \times List(Subset(R[\underline{x}])) \longrightarrow T \times R[\underline{x}]$. The third argument of $\mathrm{Simplify}$ is the list of all the already computed matrices. A complete description of this function will be given below.

Improved Algorithm $F4$

Input: $\begin{cases} F \text{ a finite subset of } R[\underline{x}] \\ \mathcal{S}el \text{ a function } List(Pairs) \rightarrow List(Pairs) \text{ such that } \mathcal{S}el(l) \neq \emptyset \text{ if } l \neq \emptyset \\ \text{Update the part of Buchberger algorithm which select the pairs to compute,} \\ \text{using the criteria like the algorithm of p. 230 in[Bec93].} \end{cases}$

Output: a finite subset of $R[\underline{x}]$.

$G := \emptyset$ and $P := \emptyset$ and $d := 0$

while $F \neq \emptyset$ **do**

  $f := first(F)$

  $F := F \backslash \{f\}$

  $(G, P) := Update(G, P, f)$

while $P \neq \emptyset$ **do**

  $d := d + 1$

  $P_d := \mathcal{S}el(P)$

  $P := P \backslash P_d$

  $L_d := Left(P_d) \cup Right(P_d)$

  $(\tilde{F}_d^+, F_d) := Reduction(L_d, G, (F_i)_{d=1,\ldots,(d-1)})$

  for $h \in \tilde{F}_d^+$ **do**

    $(G, P) := Update(G, P, h)$

return G

The new Reduction function is identical to the previous version except that there is a new argument and that it returns also the result of Symbolic Preprocessing:

Reduction

Input: $\begin{cases} L \text{ a finite subset of } T \times R[\underline{x}] \\ G \text{ a finite subset of } R[\underline{x}] \\ \mathcal{F} = (F_k)_{k=1,\ldots,(d-1)}, \text{ where } F_k \text{ is finite subset of } R[\underline{x}] \end{cases}$

Output: two finite subsets of $R[\underline{x}]$.

  $F :=$ Symbolic Preprocessing$(L, G, \mathcal{F})$

  $\tilde{F} :=$ Reduction to Row Echelon Form of F w.r.t. $<$

  $\tilde{F}^+ := \left\{ f \in \tilde{F} \mid HT(f) \notin HT(F) \right\}$

return $(\tilde{F}^+, F)$

Symbolic Preprocessing

Input: $\begin{cases} L \text{ a finite subset of } T \times R[\underline{x}] \\ G \text{ a finite subset of } R[\underline{x}] \\ \mathcal{F} = (F_k)_{k=1,\dots,(d-1)}, \text{ where } F_k \text{ is finite subset of } R[\underline{x}] \end{cases}$

Output: a finite subset of $R[\underline{x}]$.

$F := \{\text{mult}(\text{Simplify}(m, f, \mathcal{F})) \mid (m, f) \in L\}$

$Done := \text{HT}(F)$

while $\text{T}(F) \neq Done$ **do**

  m an element of $\text{T}(F) \backslash Done$

  $Done := Done \cup \{m\}$

  if m top reducible modulo G **then**

    $m = m' * \text{HT}(f)$ for some $f \in G$ and some $m' \in T$

    $F := F \cup \{\text{mult}(\text{Simplify}(m', f, \mathcal{F}))\}$

return F

Simplify

Input: $\begin{cases} t \in T \text{ a term} \\ f \in R[\underline{x}] \text{ a polynomial} \\ \mathcal{F} = (F_k)_{k=1,\dots,(d-1)}, \text{ where } F_k \text{ is finite subset of } R[\underline{x}] \end{cases}$

Output: a non evaluated product, i.e. an element of $T \times R[\underline{x}]$.

for $u \in$ list of divisors of t **do**

  if $\exists j$ $(1 \leq j < d)$ such that $(u * f) \in F_j$ **then**

    $\tilde{F}_j$ is the row echelon form of F_j w.r.t. $<$

    there exists a (unique) $p \in \tilde{F}_j^+$ such that $\text{HT}(p) = \text{HT}(u * f)$

    if $u \neq t$ **then**

      return $\text{Simplify}(\frac{t}{u}, p, \mathcal{F})$

    else

      return $(1, p)$

return (t, f)

Lemma 2.3 *If* (t', f') *is the result of* $\text{Simplify}(t, f, \mathcal{F})$ *then* $\text{HT}(t' * f') = \text{HT}(t * f)$. *Moreover if* $\tilde{\mathcal{F}}^+$ *denotes* $(\tilde{F_k^+})_{k=1,\dots,(d-1)}$ *there exist* $0 \neq \lambda \in R$, *and* $r \in R - module(\tilde{\mathcal{F}}^+ \cup \mathcal{F})$ *such that* $tf = \lambda t' f' + r$ *with* $\text{HT}(r) < \text{HT}(t * f)$.

Proof *Termination:* Simplify constructs a sequence (t_k, f_k) such that $t_0 = t$, $f_0 = f$ and $t_{k+1} < t_k$ except perhaps for the last step. T is noetherian, this implies that the algorithm stops after r_k steps.

Correctness: The first part is obvious since $\text{HT}(u_k f_k) = \text{HT}(f_{k+1})$ so that $\text{HT}(t_k f_k) = \text{HT}(\frac{t_k}{u_k} f_{k+1}) = \text{HT}(t_{k+1} f_{k+1})$. The proof is by induction on the step number. So we suppose $r_k = 1$, $t' = \frac{t}{u}$ and $u * f \in F_j$, $f' \in \tilde{F}_j$ for some j with $\text{HT}(f') = \text{HT}(u * f)$. The set $F_- = \{uf\}$ can be supplemented by other elements of F_j such that $\text{HT}(F_-) = \text{HT}(F)$ and

$size(F_{-}) = size(\mathrm{HT}(F))$. We can apply Corollary 2.1 we find $(\alpha_k) \in R$, $g_k \in F_{-} \cup (\tilde{F}_j)_+$, such that $f' = \sum_k \alpha_k g_k$ and $\mathrm{HT}(g_1) = \mathrm{HT}(f')$ and $\mathrm{HT}(f') > \mathrm{HT}(g_k)$ for $k > 2$. By construction of F_{-}, $g_1 = u * f$. Hence $f' = \alpha_1 u f + r$ with $\mathrm{HT}(r) < \mathrm{HT}(f')$, consequently $\alpha_1 \neq 0$ and we have $tf = \frac{1}{\alpha_1} t' f' - \frac{1}{\alpha_1} t' r$. $\square$

Remark 2.6 *Experimental observation establishes that the effect of* $\mathrm{Simplify}$ *is to return, in* 95% *of the cases, a product* (x_i, p) *where* x_i *is a variable (and frequently the product* (x_n, p)*). This technique is very similar to the FGLM algoritm for computing normal forms by using matrix multiplications.*

For the verification of the improved version of the algorithm we recall the following definition and theorem([Bec93], p. 219):

Definition 2.7 *Let* P *be a finite subset of* $R[\underline{x}]$*,* $0 \neq f \in R[\underline{x}]$*, and* $t \in T$*. Suppose* $f = \sum_{i=1}^{k} m_i p_i$ *with monomials* $0 \neq m_i \in R[\underline{x}]$ *and* $p_i \in P$ *not necessarily pairwise different* $(1 \leq i \leq k)$*. Then we say that this is a* t-representation *of* f *w.r.t.* P*, if* $\mathrm{HT}(m_i p_i) \leq t$ *for all* $i = 1, 2, \ldots, k$*.*

Theorem 2.3 *Let* G *be a finite subset of* $R[\underline{x}]$ *with* $0 \notin G$*. Assume that for all* $g_1, g_2 \in G$*,* $spol(g_1, g_2)$ *either equals zero or it has a* t*-representation w.r.t.* G *for some* $t < lcm(\mathrm{HT}(g_1), \mathrm{HT}(g_2))$*. Then* G *is a Gröbner basis.*

Theorem 2.4 *Let* F *be a finite subset of* $R[\underline{x}]$*,* $\mathcal{F} = (F_k)_{k=1,\ldots,(d-1)}$*, where* F_k *is finite subset of* $R[\underline{x}]$*,* $Pair(g_1, g_2) = (lcm_{1,2}, t_1, g_1, t_2, g_2)$ *with* $lcm_{1,2}, t_1, t_2 \in T$ *such that the following hold:*

(i) F_k *is the image by* mult *of a finite subset of* $T \times F$

(ii) $(\tilde{F}_k)_+ \subset G$ *for* $k = 1, \ldots, (d-1)$ *(*$\tilde{F}_k$ *being as usual the row echelon of* F_k*)*

(iii) $f_i = \mathrm{mult}(\mathrm{Simplify}(t_i, g_i, \mathcal{F}))$ *for* $i = 1, 2$*.*

(iv) $spol(f_1, f_2)$ *has a* t*-representation w.r.t.* F *with* $t < lcm(\mathrm{HT}(f_1), \mathrm{HT}(f_2))$*.*

Then the S*-polynomial* $spol(g_1, g_2)$ *has a* t'*-representation w.r.t.* F *with* $t' < lcm_{1,2}$*.*

Proof Let (t'_i, g'_i) be $Simplify(t_i, g_i, \mathcal{F})$. By Lemma 2.3 we have $\mathrm{HT}(t'_1 g'_1) = \mathrm{HT}(t_1 g_1) = lcm_{1,2} = \mathrm{HT}(t_2 g_2) = \mathrm{HT}(t'_2 g'_2)$ so that (we suppose that all the polynomial are monics):

$$\begin{aligned} spol(g'_1, g'_2) &= t'_1 g'_1 - t'_2 g'_2 \\ &= (t'_1 g'_1 - t_1 g_1) + (t_1 g_1 - t_2 g_2) + (t_2 g_2 - t'_2 g'_2) \\ &= r + spol(f_1, f_2) + r' \end{aligned}$$

with $r, r' \in Id(\tilde{\mathcal{F}}^+ \cup \mathcal{F}) \subset Id(G)$ such that $max(\mathrm{HT}(r), \mathrm{HT}(r')) < lcm_{1,2}$. Hence $spol(g_1, g_2)$ has a t'-representation for $t' = max(\mathrm{HT}(r), \mathrm{HT}(r'), t) < lcm_{1,2}$ $\square$

2.5 Selection strategy

The choice of a good selection strategy, that is to say the choice of the function $\mathcal{S}el$, is very important for the performance of the algorithm.

Computing Gröbner bases for a degree ordering is very frequently the most difficult step in the solving process (other steps are elimination or decomposition of the ideal). One reason for that is that the input of the algorithm is only a subset of $R[\underline{x}]$ with no mathematical structure. We want to give some structure to these polynomials at the beginning of the computation: we use the concept of d-gröbner bases:

Definition 2.8 *If G_d is the result of the Buchberger algorithm truncated to the degree d (that is to say we reject all critical pairs whose total degree (Definition 2.6) are $> d$), then we call G_d a (truncated) d-Gröbner basis of I.*

The following theorem give a structure to this list when the polynomials are homogenenous.

Theorem 2.5 *([Laz83], [Bec93] p. 471) For homogeneous polynomials $f_1, \ldots, f_l$, G_d is a Gröbner basis "up to degree d" that is to say:*

- $\xrightarrow[G_d]{*}$ *is well defined for polynomials f such that* $\deg(f) \leq d$.
- $\forall p \in I \; s.t. \deg(p) \leq d \implies p \xrightarrow[G_d]{*} 0$
- $spol(f,g) \xrightarrow[G_d]{*} 0$ *for f, g in G_d such that* $\deg(lcm(\mathrm{HT}(f), \mathrm{HT}(g))) \leq d$

Moreover, there exists a D_0 such that for all $d \geq D_0$, $G_d = G_{D_0}$ is the Gröbner basis of I. We note by ∞ the number D_0.

An effective Nullstellensatz may give an estimate of D_0; from a theoretical point of view such an explicit bound for the degrees reduces the problem of finding the polynomials G_∞ to the resolution of a system of linear equations. This reduction to linear algebra of the computation Gröbner bases has been used for a long time for analysing the complexity of Buchberger algorithm [Laz83].

For practical computation this does not work well since:

- D_0 is often over estimated.
- the linear system is huge: the matrix which is generated is frequently larger that really needed.
- the matrix of the linear system has a generalized sylvester structure and solving efficiently such a system is not a well known task.

Other algorithms are also closely related to linear algebra [Laz79, Laz81, Att96, Ger95, Lom98].

In fact the Buchberger algorithm and the F_4 algorithm give *incremental* methods to solve this systems. The new algorithm will compute G_{d+1} from G_d. Thus the algorithm transforms a mathematical object (G_d is unique) into another object with a stronger structure. In fact the Buchberger algorithm is also incremental since it computes one polynomial after another but in our case we compute a whole new truncated basis.

$$G_0 \longrightarrow G_1 \longrightarrow G_2 \longrightarrow \cdots \longrightarrow G_d \longrightarrow G_{d+1} \longrightarrow \cdots$$

Unfortunately Theorem 2.5 is false for non homogeneous polynomials. One solution to overcome this difficulty is to homogenize the list of polynomials but it is not efficient for big systems since parasite solutions are also computed (solutions at infinity can be of greater dimension); a better method is to consider the sugar degree [Gio91] of each polynomial: we add a "phantom" homogenization variable and the sugar degree $\deg_S$ of a polynomial is the degree of the polynomial if all the computations were performed with the additional variable:

Definition 2.9 *For the initial polynomials:* $\deg_S(f_i) = \deg(f_i)$ *, for all* $i = 1, \ldots, m$. *The polynomials occurring in the computation have the following forms* $p + q$ *or* $t * p$ *where* $t \in T$ *is a term. We define* $\deg_S(p+q) = max(\deg_S(p), \deg_S(q))$ *and* $\deg_S(m * p) = \deg(m) + \deg_S(p)$. *We say that* $\deg_S(q)$ *is the "sugar" of* q.

Definition 2.10 $G_d^{(S)}$ *is the result of the Buchberger algorithm when we reject all critical pairs whose* $\deg_S$ *is* $> d$. *(We replace* $\deg$ *by* $\deg_S$ *is the Definition 2.8)*

The weak point of this approach is that $G_{d+1}^{(S)} \backslash G_d^{(S)}$ contains polynomials with various degrees and that near to the end of the computation $\deg_S(p) \gg \deg(p)$.

We give now some possible implementation of $\mathcal{S}el$. These results are not discussed in detail as they will been reported in a more technical paper.

- The easiest way to implement $\mathcal{S}el$ is to take the identity ! In that case we really reduce all the critical pairs at the same time.
- The best function we have tested is to take all the critical pairs with a minimal total degree:

  S*el***(P)**
  Input: P a list of critical pairs
  Output: a list of critical pairs
  $d := min\{\deg(lcm(p)),\ p \in P\}$
  $P_d := \{p \in P \mid \deg(lcm(p)) = d\}$
  return P_d

  We call this strategy the *normal strategy for* F_4. If the input polynomials are homogeneous, we already have a Gröbner basis up to degree $d-1$ and $\mathcal{S}el$ selects exactly all the critical which are needed for computing a Gröbner basis up to degree d.

- We can also change the $\deg(Pair(f_i, f_j))$ to be the sugar degree, $\deg_S(lcm_{ij})$ (see Definition 2.6). In our experiments, this variant of the algorithm was less efficient.

2.6 Example

The reader should be aware that it is impossible to fully appreciate the efficiency of the algorithm for small examples. We consider the cyclic 4 problem. We choose a total degree reverse lexicographical ordering and the normal strategy.

$F = [f_4 = abcd - 1, f_3 = abc + abd + acd + bcd, f_2 = ab + bc + ad + cd, f_1 = a + b + c + d]$

At the beginning $G = \{f_4\}$ and $P_1 = \{Pair(f_3, f_4)\}$ so that $L_1 = \{(1, f_3), (b, f_4)\}$. We enter now in Symbolic Preprocessing($L_1, G, \emptyset$); $F_1 = L_1$, $Done = \mathrm{HT}(F_1) = \{ab\}$ and $T(F_1) = \{ad, ab, b^2, bc, bd, cd\}$, we pick an element in $T(F_1)\backslash Done$, for instance ad, but ad is top reducible by G; we have $Done = \{ab, ad\}$, $F_1 = F_1 \cup \{df_4\}$ and $T(F_1) = T(F_1) \cup \{d^2\}$. Since the other elements of $T(F_1)$ are not top reducible by G, Symbolic Preprocessing returns $\{f_3, bf_4, df_4\}$. Or in matrix form:

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

the row echelon form of A_1 is

$$\tilde{A}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 1 & 0 \end{bmatrix}$$

that is to say $\tilde{F}_1 = \{f_5 = ad + bd + cd + d^2, f_6 = ab + bc - bd - d^2, f_7 = b^2 + 2\,bd + d^2\}$ and since $ab, ad \in \mathrm{HT}(F_1)$ we have $\tilde{F}_{1+} = \{f_7\}$ and now $G = \{f_4, f_7\}$.

In the next step we have to study $P_2 = \{Pair(f_2, f_4)\}$, thus $L_2 = \{(1, f_2), (bc, f_4)\}$ and $\mathcal{F} = \{F_1\}$. In Symbolic Preprocessing we first try to simplify $(1, f_2)$ and (bc, f_4) with $\mathcal{F}$. We see that $bf_4 \in F_1$ and that f_6 is the unique polynomial in $\tilde{F}_1$ such that $\mathrm{HT}(f_6) = \mathrm{HT}(bf_4) = ab$, hence $\mathrm{Simplify}(bc, f_4, \mathcal{F}) = (c, f_6)$. Now $F_2 = \{f_2, cf_6\}$ and $T(F_2) = \{abc, bc^2, abd, acd, bcd, cd^2\}$. We pick abd wich is reducible by bdf_4 but again we can replace this product by bf_5. After several steps we find $F_2 = \{cf_5, df_7, bf_5, f_2, cf_6\}$

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

$$\tilde{A}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & -1 & 0 \end{bmatrix}$$

$\tilde{F}_2 = \{f_9 = acd + bcd + c^2d + cd^2,\ f_{10} = b^2d + 2\,bd^2 + d^3,\ f_{11} = abd + bcd - bd^2 - d^3,$ $f_{12} = abc - bcd - c^2d + bd^2 - cd^2 + d^3,\ f_{13} = bc^2 + c^2d - bd^2 - d^3\}$ and $G = \{f_4, f_7, f_{13}\}$.

In the next step we have $L_3 = \{(1, f_1), (bcd, f_4), (c^2, f_7), (b, f_{13})\}$, and we call recursively the function Simplify: $Simplify(bcd, f_4) = Simplify(cd, f_6) = Simplify(d, f_{12}) = (d, f_{12})$. We have $F_3 = \{f_1, df_{12}, c^2 f_7, bf_{13}\}$. Notice that $c^2 f_7$ cannot be simplified, but very often we have only a polynomial multiplied by a variable. After several steps in Symbolic Preprocessing we find $F_3 = \{f_1, df_{12}, c^2 f_7, bf_{13}, df_{13}, df_{10}\}$ and $\tilde{F}_3 = \{f_{15} = c^2b^2 - c^2d^2 + 2\,bd^3 + 2\,d^4,\ f_{16} = abcd - 1,$ $f_{17} = -bcd^2 - c^2d^2 + bd^3 - cd^3 + d^4 + 1,\ f_{18} = c^2bd + c^2d^2 - bd^3 - d^4,\ f_{19} = b^2d^2 + 2\,bd^3 + d^4\}$. Note that the rank of F_3 is only 5. This means that there is a reduction to zero.

2.7 Conclusion

So we have transformed the degree of freedom in the Buchberger algorithm into strategies for efficiently solving linear algebra systems. This is easier because we have constructed a matrix A (the number of rows in A is a little overestimated by the symbolic prediction) and we can decide to begin the reduction of one row before another with a *"good reason"*. For integer coefficients it is a major advantage to be able to apply an iterative algorithm on the whole matrix (p-adic method). Some negative aspects are that the matrix A is singular and that A is often huge. A good compression algorithm for the matrix reduces the storage requirements by a factor greater than 10 (see Section 3.4).

3 Solving sparse linear systems

Compared with naive linear algebra approach we have reduced dramatically the size of matrices which are involved. Despite this reduction, the matrices that we have to solve during the program execution are very big sparse matrices (see p. 22): say 50000×50000 to give an idea (the record is 750000×750000 for a very sparse matrix). To give a comprehensive review of all useful linear techniques is far beyond the scope of this paper and we give only some references to the original papers. It should be observed that we have to solve *sparse* matrices in a *computer algebra* system. It is therefore not surprising that we have to merge techniques coming from sparse linear algebra (possibly designed initially for floating point coefficients) and techniques coming from computer algebra (for big integer computations for instance). But first we establish the link with the main algorithm:

3.1 Solving a matrix and reduction to row echelon

In the main algorithm we have to *reduce* (find a basis of the image of the corresponfing linear map) sparse matrices which are singulars and not square. On the other hand, linear algebra techniques are often described for *solving* linear systems of equations $Ax = b$. One way to connect the two approaches is to first extract a square sub matrix and to put the remaining columns into the right hand side. For instance if we want to reduce:

$$A = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccc} a^2 & ab & b^2 & bc \end{array} \\ \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 2 & 2 \end{pmatrix} \end{array}$$

We first try to find the rank of the matrix using a fast algorithm mod p and we find that the second column is deffective. So we extract a square matrix and a right hand side:

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \end{bmatrix} \quad r = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

Hence the system of equation can be rewritten:

$$\tilde{A} \begin{bmatrix} a^2 \\ b^2 \\ bc \end{bmatrix} = -(ab)r \ \ or \ \ \begin{bmatrix} a^2 \\ b^2 \\ bc \end{bmatrix} = -(ab)(\tilde{A}^{-1}r)$$

so we have to *solve* the linear system

$$\begin{bmatrix} x_0 \\ z_0 \\ t_0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

and the reduced form of the matrix

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccc} a^2 & ab & b^2 & bc \end{array} \\ \begin{pmatrix} 1 & x_0 & 0 & 0 \\ 1 & z_0 & 1 & 0 \\ 1 & t_0 & 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccc} a^2 & ab & b^2 & bc \end{array} \\ \begin{pmatrix} 1 & 2 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

3.2 Solving sparse linear systems

Solving sparse linear systems has been studied for a very long time for numerical computations, there are mainly two types of methods: iterative methods (computing successively $y_{i+1} = Ay_i$ if A is a $n \times n$ matrix) and decomposition methods. The methods wich are useful in our context are the following:

3.2.1 Iterative solvers

After $O(n)$ steps we obtain an exact solution (up to rounding errors). This is the case of conjugate gradient method or the Lanczos algorithm [Mon95] since after n steps the result is exact. Another well known iterative algorithm is the Wiedemann algorithm [Wie86, Kal91], which uses the efficient Berlekamp and Massey algorithm [L.69] to recover the solution. Note that there exists a more efficient version of the Wiedemann algorithm: the Wiedemann algorithm by blocks (but we have not implemented this version yet).

The key operation in these algorithms is the multiplication $A * y$. It is very easy to implement this operation efficiently to take advantage of the sparsity of A and to obtain a complexity of $O(|A|)$ for computing $A * y$ where $|A|$ is the number of non zero elements in A; thus the global complexity for solving the system is $O(n|A|)$ instead of $O(n^3)$.

When the matrix has a regular pattern it is possible to apply even more efficient techniques (for Toeplitz matrices for instance [Bin94]). These methods can not be applied in our case since the pattern of the generated matrices is not regular enough. A significant drawback of these methods is that there is no speedup if we try to solve simultaneously several linear systems with the same left hand side.

3.2.2 Factorization methods

The classical LU decomposition try to decompose the input matrix A into a product LU where L (resp. U) is a lower (resp. upper) triangular matrix. For *sparse* LU decomposition [Geo81, Rei71, Ros72, Duf84, Geo81, Geo93] there is an additional constraint: the number of non zero elements in L and U minus the number of non zero elements in A must be as smallest as possible (this number is called the number of "fill-ins"). The sparse LU decomposition starts with a symbolic preprocessing very similar to our. It aims is to avoid to spend time time for computing coefficients the vanishing of which is predictable. The interest of this method is that both preprocessing may be done simultaneously.

A large number of implementation for these methods are available (mainly in C/C++ or Fortran) [Dav95] or even in Matlab. Unfortunately these programs and some algorithms are not very robust: very often the input matrix A must be non singular or square or positive definite. On the other hand, parallel algorithms and parallel implementations exist [Van93, Hea, Don, Alv, Pey]. We have modified the smms [Alv93] program in order to work with modulo p coefficients in order to evaluate the costs of different algorithms.

Solving large linear equations modulo a small prime number has been studied in cryptology [LaM91, Mon95] for factoring integers or computing discrete logarithms (very often $p = 2$ in that case). These authors use a combination of structured Gaussian elimination and other iterative algorithms. In the current implementation of the F_4 algorithm we use by default the structured Gaussian elimination. Note that in an ideal implementation the program should first analysed the shape of the matrix and then decided which algorithm should be applied.

3.3 Computer Algebra methods for solving linear systems with integer coefficients.

Very specific methods have been developed in Computer Algebra for solving linear equations when the coefficients are integers. First we recall that the Bareiss algorithm is better than the classical Gauss algorithm but is very inefficient for big systems.

The best way of solving linear systems $Ax = b$ with integer coefficients is to use p-adic computations: we choose a prime p, and we compute $C = A^{-1} \, mod \, p$ (very often a sparse LU decomposition is more appropriate). Then we define

$$\begin{cases} b^{(0)} = b \\ y^{(m)} = Cb^{(m)} \, mod \, p \\ b^{(m+1)} = \frac{b^{(m)} - Ay^{(m)}}{p} \\ x^{(m)} = \sum_{j=0}^{m-1} y^{(j)} p^j \end{cases}$$

Theorem 3.1 *(Dixon [Dix82])*

- $Ax^{(m)} - b = 0 \, mod \, p^m$
- *If B is upper bound of the coefficients of A and b it is sufficient to compute $x^{(m)}$ for $m \geq 2n \log_p(nB)$ and then to apply the extended Euclidean algorithm to $x^{(m)}$ and p^m to recover the value in* **Q**.

In fact the iteration may be stopped when $x^{(m)}$ becomes stable (see [S.R71] for multi modular methods which may be generalized to $p-$adic method).

The global complexity [Knu81] of this algorithms is

$O(n^5 log(nB)^2)$	Bareiss method
$O(n^3(n + log(nB))log(nB))$	multi modular method
$O(n^3 log(nB)^2)$	p-adic method

Note that the p-adic method is also an iterative algorithm (in fact this is a Newton algorithm) and that we have previously noticed that this kind of algorithm is less efficient for solving simultaneous systems $Ax = b_1, \cdots, Ax_k = b_k$. If $k \geq n$ it is probably better to use a multi modular approach: we compute $A^{-1}b \, mod \, p_i$ for different primes p_i and we apply the Chinese remainder theorem to find the solution modulo $\Pi_i p_i$.

3.4 Matrix compression

When the matrices are big it is necessary to adopt fairly complicated storage schemes to compress the matrix in memory: consider a $5.10^4 \times 5.10^4$ matrix with 10% non zero elements (this is the case in Cyclic 9 for instance); even if only one byte is allocated for each coefficients (an optimistic assumption since some coefficients have hundred digits); the matrix requires $250 * 10^6$ bytes to be stored ! In our implementations we do not duplicate coefficients (relying on the fact that some rows in the matrix are just term multiples of others); thus we have only to compress the position of the non zero elements; we have experimented the following techniques:

(i) No compression: inefficient both for CPU time and memory usage.

(ii) Bitmap compression: each row of the matrix contains zero and non zero elements:

```
x 0 0 x x 0 0 0 x ...
```

$j_1, j_2, \ldots$ denote the indexes of the non zero elements (here $j_1 = 1, j_2 = 4, \ldots$), then $\sum_k 2^{j_k - 1}$ is the bitmap compression (for the example $1 + 2^3 + 2^8 = 281$) . This method is efficient but consumes a substantial amount of memory. This is the prefered way to implement the compression when the coefficients are in $GF(2)$

(iii) Distributed representation: the row is represented by the array

$$\boxed{j_1}\boxed{j_2 - j_1}\boxed{j_3 - j_2}\cdots$$

using bytes when $j_k - j_{k-1} < 255$ (this occurs very frequently). This method is most efficient in memory than the previous one and a little slower (10%).

(iv) Apply a standard compression tool (gzip for instance) to one of the previous representation. Very efficient in memory but rather slow.

For the moment our prefered methods are (ii) and (iii) (depending on the ground field), but the compression algorithms should be seriously improved in future versions.

4 Experiments

The quality of the computer implementation of Gröbner bases computations may have a dramatic effect on their performance. The theoretical results on complexity (even for homogeneous systems, the complexity is d^{3n} in most cases and 2^{2^n} in some very pathological cases) cannot throw light on the practical behavior of effective implementation. Thus, while "paper and pencil" analysis of polynomial solving algorithms are important, they are not enough. Our view is that studying and using programs which implement these algorithms is an essential component in a good understanding of this area of Computer Algebra. To this end, we provide some experiments and comparison with similar programs. This section should be considered as the validation of our algorithm. Thus it plays the same role as its proof for theorems.

4.1 Methodology

Empirical analysis means that we have to pay particular attention to the development of methodologies in the field of benchmark for linear system solving. We adopt the following points:

1. We compare the new algorithm with *state of the art* Gröbner bases implementation (namely Magma [Can98], PoSSo/Frisco Gröbner engine [Tra97], Macaulay 2 [Sti89, Gra97], Singular [Gre97], Asir [Tak96], Cocoa, Axiom, Maple [Cha91] and Gb [Faub]). It is also crucial to compare the implementation of the new algorithm with the Buchberger algorithm implemented by *the same person* (in this case the author). In our opinion it is also important to compare low-level implementation of the algorithms to avoid parasite interactions with a high level language/compiler. F_4 has been implemented in C and most of the competitors are implemented in C/C++.

2. The list of examples is also a crucial issue: the examples can be *easily accessed* [Faua] (the web site contains pointers to the Frisco test suite). The list is composed of classical benchmarks (Cyclic n, Katsura n) but also of industrial examples (coming from signal theory, robotics). We have removed from the list all the *toy examples* since nothing can be concluded with them. Of course the toy concept is relative to current computers. For us, an example is toy if it takes less than 1 sec on a PC.

3. This section contains two timing tables: the first one corresponds to modular computations, the second one corresponds to big integers computations. All the computations are done several times on equivalent computers to prevent as much as possible interactions with other programs. For each timing the program was run several times. This was necessary to eliminate fluctuations in the measurements stemming from some other programs running on the same computer. Of course the timings are rounded.

4. We rigorously use the last version of all the programs and use an appropriate computer to execute them.

5. An even more convincing proof of the efficiency of a new algorithm is to solve previously untractable problems. So we should test the algorithm on very difficult problems. In our case, F_4 solves Cyclic8 and Cyclic9.

6. The same strategy is used for all the programs. For instance if we homogenize the input polynomials for one program we try the same strategy with all other programs. This is the reason why we give the timing for cyclic n and homogeneous cyclic n. Some systems (Singular for instance) allow the user to customize the internal strategy; we try several parameters and we retain the best method.

7. The outputs of the different programs are checked to be equal.

8. Last, and it is the more difficult, we compare the algorithm with other methods: triangular sets (Moreno/Aubry), homotopies (J. Verschelde), Bezoutian (Mourrain) and sparse resultant (Emiris). The task is more difficult since the outputs are very different (quasi component, floating point approximations).

    - Triangular methods (Wu, Wang, Lazard, Kalkebrenner). On one hand triangular methods seem to be less efficient than lexico Gröbner bases computation (the current limit is Cyclic 6 and Katsura 5) but on the other hand the quality of the output is better. So we think that these methods are useful to simplify lexicographical Gröbner bases.

- Homotopy methods. J. Vershelde reports timings (Cyclic 8 on a Sparc-Server 1000 in $4h35m$) which are less efficient than Gröbner bases techniques. It is difficult to handle over constrained systems.
- Mixed volume is extremely fast to estimate the number of isolated roots (up to Cyclic' 12) but with our experience it is not so efficient for other systems (and the number of solution is often over estimated).

4.2 Modular Computations

Modular computations are very useful in Computer Algebra because they give a fast result (with a very high probability) and information on the number of solutions (Hilbert function).

Moreover, since big integer computations could be done by means of p-adic or multi modular arithmetics it means that the cost of an integer computation is roughly

$$\text{time of modular computation} * \text{size of the output coeffs}$$

So it is also very important to have an efficient solver modulo a prime p.

The computer is a *PC Pentium Pro 200 Mhz with 128 Meg of Ram.*

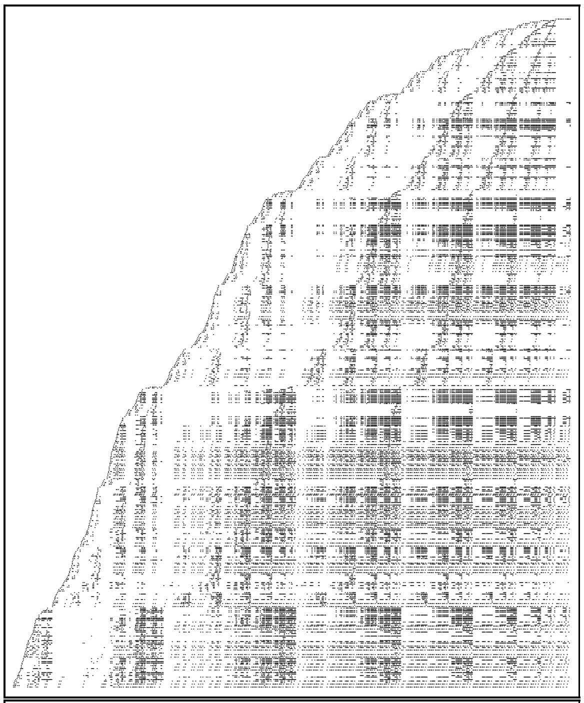
We consider only non homogeneous systems, but in the following we give $\tau = \min(\tau_{\text{homog}}, \tau_{\text{non homog}})$ because Singular and Macaulay are often very slow for non homogeneous system.

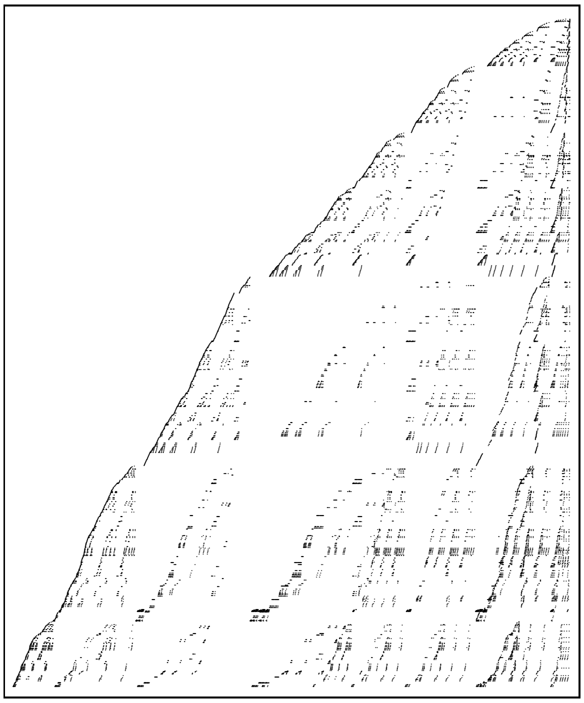
Note that the PoSSo library uses generic coefficient to implement Z/pZ but the other softwares implement inlined modular arithmetic. In other words the overhead of function calls is heavier in the PoSSo library. With a better implementation one can probably divide timings by a factor between 2 and 4.

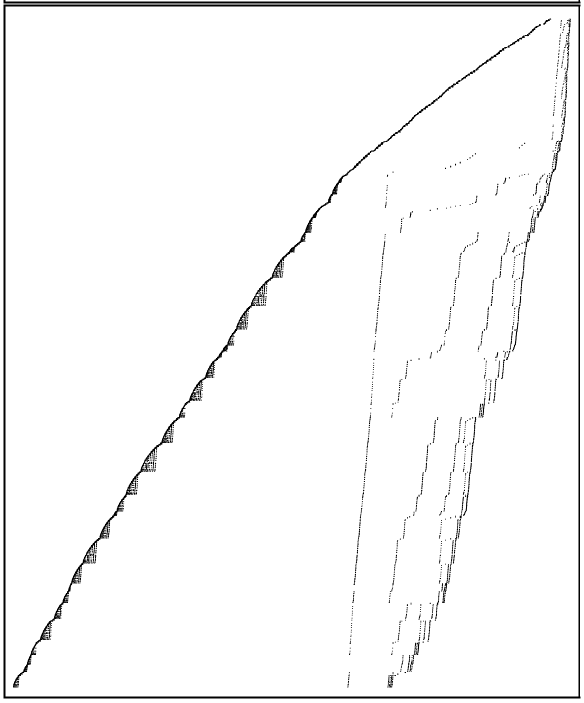
Table 1 *Academic Examples mod* p

	FGb	*Gb*	*PoSSo*	*Singular*	*Mac 2*	*Asir*
K_7	*0.8 s*	*3.2 s*	*46 s*	*10 s*	*12 s*	*56 s*
K_8	*4.1 s*	*29 s*	*9m58s*	*1m47s*	*1m55s*	
K_9	*30 s*	*3m48s*	*1h25m*	*17m41s*	*17m11s*	
K_{10}	*4m13s*	*36m23s*		*3h6m*		∞
K_{11}	*30m29s*		∞	∞	∞	∞
Cyclic 7	*4.6 s*	*1m15s*	*9m3s*	*2m34s*	*2m0s*	*6m51s*
Hom C_7	*5.2 s*	*1m02s*				
Cyclic 8	*1m55s*	*26m17s*	*4h38m*	*1 h 56 m*	*1 h 33m*	*3h54m*
Hom C_8	*184.4s*	*39m17s*				
Cyclic 9	*4 h 32 m*	∞	∞	∞	∞	∞
Hom C_9	*11 h 10 m* [a]	∞	∞	∞	∞	∞

[a] 11 h 9 min 40 s on a 500 Mhz Alpha workstation with 1Go of RAM.







We give the shape of some generated matrices. The matrices can have a very different structure and the number of non zero elements varies greatly.

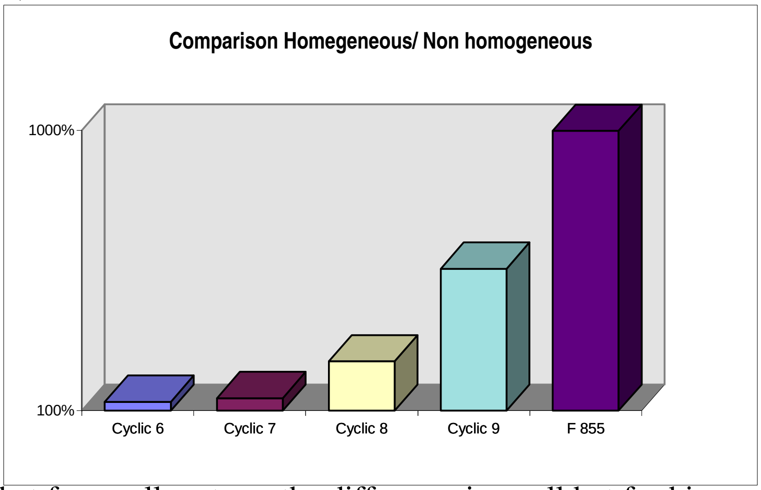
Upper left figure:
From $Cyclic_7$
Matrix A_9 475×786 (13.8% non zero)

Upper right figure:
application example: from $f_{8,5,5}$ [Fau98b]
matrix A_5 1425×2561 (0.47% non zero)

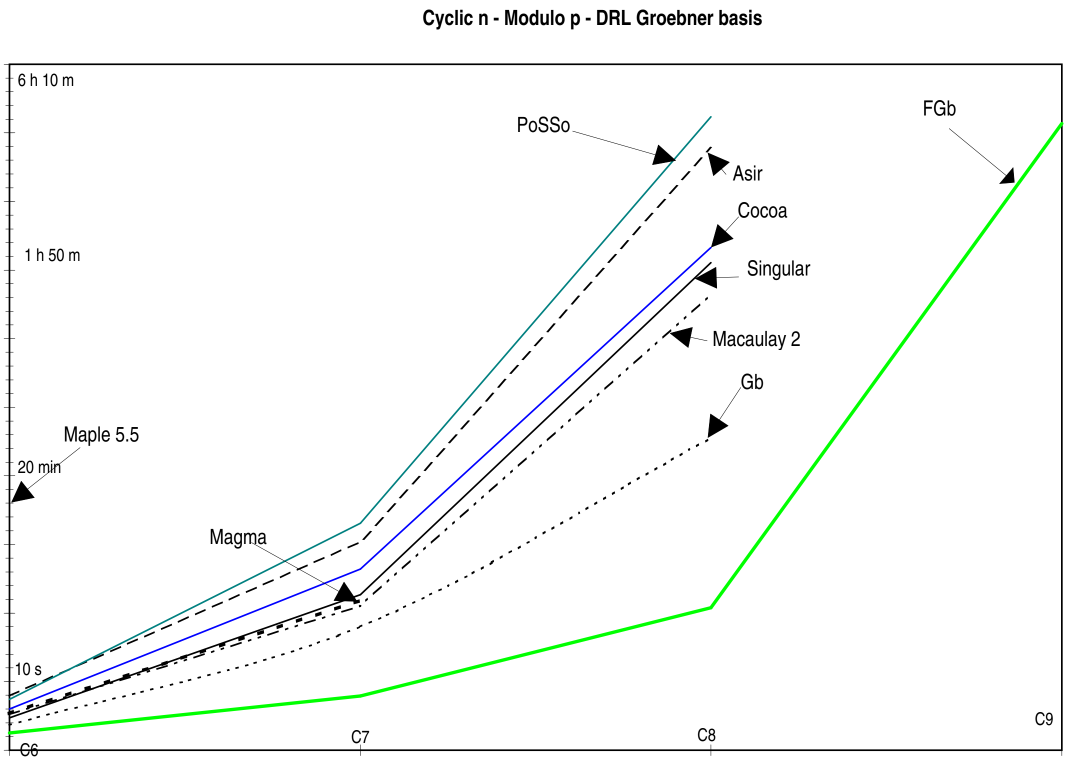
Lower left figure:
Example: engineering problem (Nuclear)
matrix A_7 1557×3838 (0.2% non zero)

The conclusion of Table 1 is that *the old Gb is still faster than other systems, and that FGb* $\gg$ *Gb*.

The following figure shows the performance results for the homogenization strategy and the affine strategy (the quotient "time to solve homog Cyclic n" divided "time to solve non homog Cyclic n" is plotted).



We conclude that for small systems the difference is small but for big systems like Cyclic 9 there is a huge difference (you add several components of dimension 3 and 4). For Cyclic 9 the algorithm generates a 292051×317850 matrix and it is 3.2 times slower!



4.3 Integers

In the following table, we have included a special version of the the classical PoSSo Gröbner engine called "Rouil", this version has been optimized by Fabrice Rouillier[2].

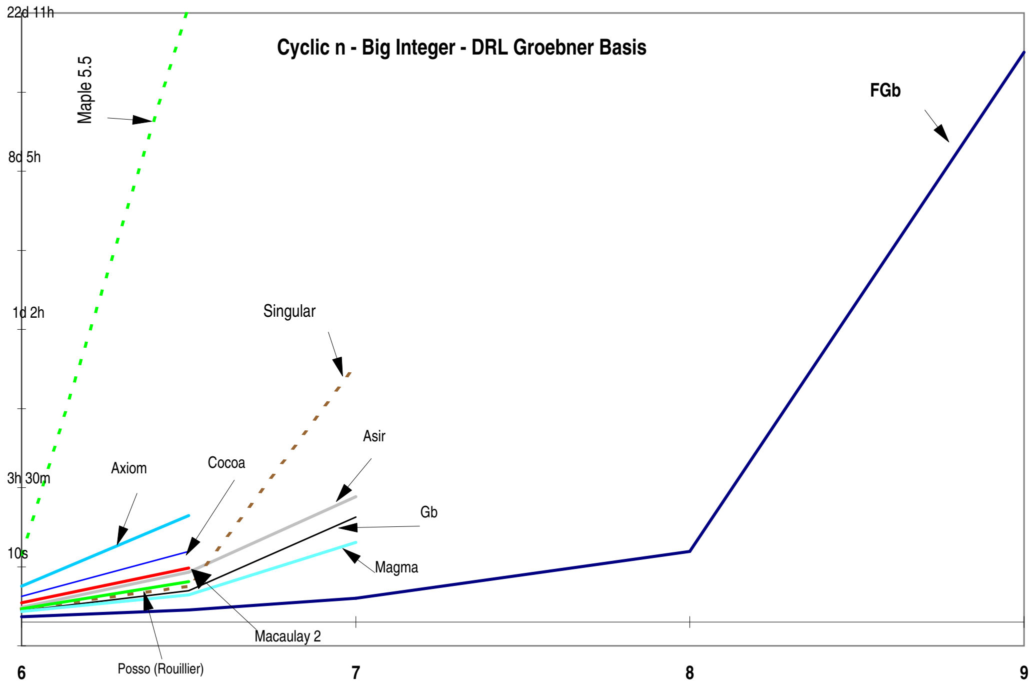
In the table we remove the Singular entry because Singular and Macaulay2 seem very close and very slow for big integers.

`Size` is the size in digits of the biggest coefficient in the output.

Table 2 *Academic Examples* $\mathbf{Z}$

	FGb	*Gb*	*Singular*	*Rouil*	*Asir*	*Mac 2*	*Magma*	*Size*
K_7	*2.9 s*	*42 s*		*50 s*	*3m28s*	*10m56s*		*53*
K_8	*23 s*	*10m21s*		*8m23s*	*8m37s*	*2h17m*		*102*
K_9	*3m3s*	*5h35m*		*1h31m*		∞		*133*
K_{10}	*31m24s*	∞		∞	∞	∞		*192*
Cyclic 6	*0.3 s*	*3.2 s*	*5.3 s*	*5.4 s*	*8 s*	*19.5 s*	*2.6 s*	*96*
Hom C_7	*54.2s*	*1h32m*	*10h35m*	*>25m*[a]	*2h50m*		*2202s*	*96*
Cyclic 7	*39.7s*	*5h17m*[b]	∞	∞	*>2h*[c]	∞	=	*96*
Cyclic 8	*24m4s*	∞	∞	∞	∞	∞	∞	*202*
Cyclic 9	*18 days*[d]	∞	∞	∞	∞	∞	∞	*800*

[a] 134 Mbyte of data when stopped.
[b] Estimated from an original time of 24h26m40s on Sparc 10 with Lazard s algorithm.
[c] 162 Mbyte of data when stopped.
[d] The size of the result in binary is 1 660 358 088 bytes. Run on 512 Meg RAM PC.



[2] The algorithm uses a prime p to avoid syzygies. Then the algorithm checks that the result was correct. At the present state the implementation sometimes does not detect bad primes.

We observe that Magma is very efficient for big integers (in fact the Magma version for alpha workstations is even 7 times faster).

Cyclic-9 for big integers is an example of *huge* computation; we use:

- 3 Processors PPro 200 Mhz 512 Meg RAM + 1 Alpha 400 Mhz 570 Meg
- Total sequential CPU time: *18 days*
- Size of the file of coefficients in the output (binary): *1660 Meg*
- The result contains: 1344 polynomials with 1000 monomials and 800 digits numbers !

This success is also a failure in some sense: the size of the output is so big that we cannot do anything with this result. That is to say we are now near to the intrinsic complexity of Gröbner bases. On the other side, the output is very big because the coefficients are big and floating point computation would not suffer from this exponential growth.

We conclude that for all these examples FGb $\gg$ Gb and that it faster than other systems.

5 Final remarks

The conclusion is that F_4 is at least one order of magnitude faster than all previously implemented algorithms.

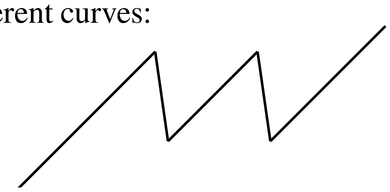
We recall the main features of this algorithm are:

- Replace all the previous strategies (sugar([Gio91]), $\ldots$) by algorithms for (sparse) linear algebra. It explains why the usual strategies in Buchberger algorithm could not be optimal.
- Faster for all kind of arithmetic (modular computation, integers, "generic" computation)
- In some sense it is as fast as possible for big integers coefficients or coefficients with parameters ($k(y_1, \ldots, y_r)$) because the practical complexity is almost linear in the size of the output coefficients: in the case of homogeneous polynomials the complexity of F_4 is $s^{1+o(1)} + M$ where s is the size of othe result and M is the time needed for modular computation which is generally much smaller.
- For homogeneous systems the algorithm generate reduction to zero or non zero polynomials (completely reduced) which are all in the final Gröbner basis. So the algorithm does not generate "parasite" intermediate data.
- Very good experimental behavior for non homogeneous systems (several times faster than the corresponding homogeneous system).
- Parallel versions of the algorithm can be implemented (we have done a first implementation).

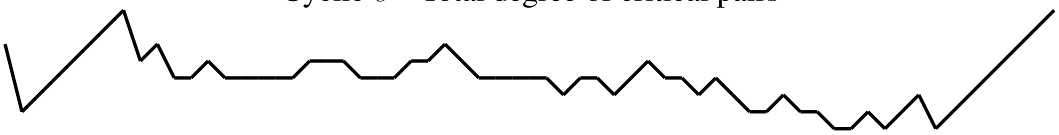
- The algorithm is easier to implement (no polynomial arithmetic is required, do not need an efficient power product (exponents) implementation).

- It can solve previously intractable problems: we are now able to compute easily three new records: Cyclic-9 for modular coefficients (4h30), Cyclic-8 for big integers coefficients (25 mins) and the very challenging Cyclic-9 for bignum coefficients (18 days of CPU time).

A lot of work remains to do in the linear algebra part to apply less naive algorithms to the generated matrices and to compress more efficiently those matrices. Considerable works must also be done to compare the algorithm with different possible strategies (sugar and homogenizing, multi-weight in the case of multi-homogeneous ideals might reduce the size ot the matrices (as suggest by D. Lazard and one anonymous referee)). How to use the symmetry of the problems to handle more efficiently the matrices is also an open problem. Even if the algorithm presented in this paper is heavily connected with the Buchberger algoritm (use the same criteria for useless pairs), we think that an interesting work would be to use Mandache [Man94, Man96]'s technique to check that F_4 *is not* a Buchberger algorithm in the sense that the Buchberger algorithm cannot simulate the new algorithm for any strategy. When the normal strategy is used, we can plot the function $d \longrightarrow deg(first(P_d))$; we obtain an increasing function for homogeneous systems but in the affine case we obtain different curves:



Cyclic 8 – Total degree of critical pairs



El 50 – Total degree of critical pairs

An open issue is to understand deeply the shape of this curves.

Acknowledgements

We would like to thank Daniel Lazard and one anoymous referee for their comments on drafts of this paper. I also want to acknowledge M. Stillman and D. Bayer for helpful suggestions and encouragement. We would like to thank the UMS 658 Medicis.

References

[Alv] Alvarado F.L. and Pothen A. and Schreiber R. Highly Parallel Sparse Triangular Solution. preprint.

[Alv93] Alvarado F.L. *The Sparse Matrix Manipulation System User and Reference Manual*. University of Wisconsin, May 1993.

[Att96] Attardi G. and Traverso C. Homogeneous Parallel Algorithm. *Journal of Symbolic Computation*, 1996.

[Bec93] Becker T. and Weispfenning V. *Groebner Bases, a Computationnal Approach to Commutative Algebra*. Graduate Texts in Mathematics. Springer-Verlag, 1993.

[Bin94] Bini D. and Pan V. *Polynomial and Matrix Computations*, volume 1 of *Progress in theoretical computer science*. Birkäuser, 1994. Fundamental Algorithms.

[Buc65] Buchberger B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Innsbruck, 1965.

[Buc70] Buchberger B. An Algorithmical Criterion for the Solvability of Algebraic Systems. *Aequationes Mathematicae*, 4(3):374–383, 1970. (German).

[Buc79] Buchberger B. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Basis. In *Proc. EUROSAM 79*, volume 72 of *Lect. Notes in Comp. Sci.*, pages 3–21. Springer Verlag, 1979.

[Buc85] Buchberger B. Gröbner Bases : an Algorithmic Method in Polynomial Ideal Theory. In Reidel, editor, *Recent trends in multidimensional system theory*. Bose, 1985.

[Can98] Cannon J. *The Magma Computational Algebra System 2.20-7*, Feb 1998. `http://www.maths.usyd.edu.au:8000/u/magma/`.

[Cha91] Char B. and Geddes K. and Gonnet G. and Leong B. and Monagan M. and Watt S. *Maple V Library Reference Manual*. Spinger-Verlag, 1991. Third Printing, 1993.

[Dav95] Davis T. *Users' Guide for the Unsymmetric-pattern MultiFrontal Package*. Computer and Information Sciences Department University of Florida, January 1995. Technical Report TR-95-004.

[Dix82] Dixon J.D. Exact solution of linear equations using p-adic expansions. *Numer.Math.*, 40:137–141, 1982.

[Don] Dongarra J. and Lumsdaine A. and Niu X. and Pozo R. and Remington K. A Sparse matrix Library in C++ for High Performance Architectures. preprint.

[Duf84] Duff I.S. and Reid J.K. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Statist. Comput.*, 5(3):633–641, 1984.

[Faua] Faugère J.C. *Benchmarks for polynomial solvers.* avalaible on the WEB `http://posso.lip6.fr/~jcf/benchs.html`.

[Faub] Faugère J.C. *On line documentation of Gb.* avalaible on the WEB `http://posso.lip6.fr/~jcf`.

[Fau98a] Faugère J.C. Computing Gröbner Basis without reduction to zero (F5). Technical report, LIP6 report, 1998. in preparation.

[Fau98b] Faugère J.C. and Moreau de Saint-Martin F. and Rouillier F. Design of nonseparable bidimensional wavelets and filter banks using Gröbner bases techniques. *IEEE SP Transactions on Signal Processing*, 46(4), 1998. Special Issue on Theory and Applications of Filter Banks and Wavelets.

[Fro96] Fronville A. *Singularité résiduelle et Problème du centre*. PhD thesis, Université Paris 6, Décembre 1996.

[Geo81] George A. and Liu J. W. H. *Computer solution of large sparse positive definite systems*. Englewood Cliffs, N.J. : Prentice-Hall, 1981.

[Geo93] George A. and Gilbert J.R. and Liu J. W. H. *Graph theory and sparse matrix computation*. New York : Springer-Verlag, 1993.

[Ger95] Gerdt V.P. Involutive Polynomial Bases. In *PoSSo on software*. Paris, F, 1995.

[Gio91] Giovini A. and Mora T. and Niesi G. and Robbiano L. and Traverso C. One sugar cube, please, or Selection strategies in the Buchberger Algorithm. In S. M. Watt, editor, *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*. ISSAC' 91, ACM Press, 1991.

[GM88] R. Gebauer and H.M. Möller. On an Installation of Buchberger's Algorithm. *Journal of Symbolic Computation*, 6(2 and 3):275–286, October/December 1988.

[Gra97] Grayson D. and Stillman M. *Macaulay 2 User Manual*, 1997. `http://www.math.uiuc.edu/Macaulay2/`.

[Gre97] Greuel G.-M. and Pfister G. and Schoenemann H. *SINGULAR 1.0.2*, July 1997. `http://www.mathematik.uni-kl.de/~zca/Singular/Welcome.html`.

[Hea] Heath M.T. and Raghavan P. Distributed Solution of Sparse Linear Systems. preprint.

[Kal91] Kaltofen E. On Wiedemann's Method of Solving Sparse Linear Systems. *LNCS*, 539:29–38, 1991. AAECC-9.

[Knu81] Knuth D.E. *The Art of Computer Programming*, volume 2. Addison Wesley, 2 edition, 1981.

[L.69] Massey J. L. Shift-register synthesis and bch decoding. *IEEE Trans. on Information Theory*, 15(1):122–127, January 1969.

[LaM91] LaMacchia B.A. and Odlyzko A.M. Solving Large Sparse Linear Systems Over Finite Fields. *LNCS*, 537, 1991. Advances in Cryptology, CRYPTO'90.

[Laz79] Lazard D. Systems of Algebraic Equations. In *EUROSAM 79*, pages 88–94, 1979.

[Laz81] Lazard D. Resolution des systemes d'equations algebriques. *Theor. Comp. Science*, 15:77–110, 1981.

[Laz83] Lazard D. Gaussian Elimination and Resolution of Systems of Algebraic Equations. In *Proc. EUROCAL 83*, volume 162 of *Lect. Notes in Comp. Sci*, pages 146–157, 1983.

[Lom98] Lombardi H. Un nouvel algorithme de calcul d'une Base de Gröbner. Technical report, Equipe de Mathématiques de Besançon, Janvier 1998.

[Man94] Mandache A.M. . The Gröbner Basis Algorithm and Subresultant Theory. In *Proceedings of ISSAC'94*, pages 123–128. Oxford, UK, ACM press, July 1994.

[Man96] Mandache A.M. A note on the relationship between involutive bases and Gröbner bases . *Mega 96*, 1996.

[Mon95] Montgomery P. A Block Lanczos Algorithm for Finding Dependencies over GF(2). *LNCS*, pages 106–120, May 1995.

[Pey] Peyton B.W. and Pothen A. and Yuan X. Partitioning a Choral Graph into Transitive Subgraphs for Parlallel Sparse Triangular Solution. preprint.

[Rei71] Reid J.K. *Large Sparse Sets of Linear Equations*. Academic Press . London and New York, 1971.

[Ros72] Rose D. and Willoughby R.A. *Sparse Matrices and their applications*. Plenum Press, 1972.

[S.R71] S.R. Petrick, editor. *Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation*. ACM Press, March 23-25 1971.

[Sti89] Stillman M. and Bayer D. *Macaulay User Manual*, 1989. available via anonymous ftp on `zariski.harvard.edu`.

[Tak96] Takeshima T. *Risa/Asir*. FUJITSU LABORATORIES LIMITED, 1996. Vers 950831. available from ftp://endeavor.fujitsu.co.jp/pub/isis/asir.

[Tra97] Traverso C. *Groebner Engine. Rel. 2.0*, 1997. `http://janet.dm.unipi.it/posso\_demo.html`.

[Van93] Van der Stappen A.F. and Bisseling R.H. and Van de Vorst J.G.G. Parallel Sparse LU Decomposition on a mesh network of transputers. *SIAM J. Matrix ANAL. APPL.*, 14(3):853–879, July 1993.

[Wie86] Wiedemann D. Solving Sparse Linear Equation Over Finite Fields. *IEEE Transaction on Information Theory*, IT-32(1), January 1986.