UNIVERSITÄT KAISERSLAUTERN
Zentrum für Computeralgebra

REPORTS ON COMPUTER ALGEBRA
NO. 35

Slimgb: Gröbner Bases with Slim Polynomials

by

M. Brickenstein

September, 2005

The Zentrum für Computeralgebra (Centre for Computer Algebra) at the University of Kaiserslautern was founded in June 1993 by the Ministerium für Wissenschaft und Weiterbildung in Rheinland-Pfalz (Ministry of Science and Education of the state of Rheinland-Pfalz). The centre is a scientific institution of the departments of **Mathematics, Computer Science, and Electrical Engineering** at the University of Kaiserslautern.

The goals of the centre are to advance and to support the use of Computer Algebra in industry, research, and teaching. More concrete goals of the centre include

- the development, integration, and use of software for Computer Algebra
- the development of curricula in Computer Algebra under special consideration of interdisciplinary aspects
- the realisation of seminars about Computer Algebra
- the cooperation with other centres and institutions which have similar goals

The present coordinator of the Reports on Computer Algebra is:
Christoph Lossen (email: `lossen@mathematik.uni-kl.de`)

Zentrum für Computeralgebra
c/o Prof. Dr. G.-M. Greuel, FB Mathematik
Erwin-Schrödinger-Strasse
D-67663 Kaiserslautern; Germany
Phone: 49 - 631/205-2850 Fax: 49 - 631/205-5052
email: `greuel@mathematik.uni-kl.de`

Slimgb: Gröbner Bases with Slim Polynomials

Michael Brickenstein
Department of Mathematics
University of Kaiserslautern
Germany
bricken@mathematik.uni-kl.de

Abstract

This paper introduces a variation of Buchbergers's algorithm (Buchberger, B. (1965)) to compute Gröbner bases. It is designed to keep coefficients small and polynomials short in intermediate computations. This pays off in computation time as well as memory usage.

The algorithm is very flexible, the strategy is controlled by a single function which calculates the weighted length of a polynomial. All components of the algorithm depend on this weighted length function, so changing this one can easily adjust the whole algorithm to special problems.

While some ideas of Faugère's F4 (Faugère, J.-C. (1999)) are used, the algorithm is easier to implement in a standard Gröbner basis environment, as usual polynomial data structures are used. In fact, the algorithm does not depend on any data structures, but is compatible with all important polynomial representations, for example Geobuckets (Yan (1998)).

Key features of the algorithm are parallel reductions, weighted length strategies, exchanging generating system members for better (shorter) intermediate results and an extended version of the product criterion.

Introduction

Gröbner basis computations have made considerable progresses in the sense of better data structures, highly optimized implementations, good critical pair criteria and indirect algorithms like the Gröbner walk and the Hilbert driven Buchberger.

The present paper focusses on other aspects.

The aim is to reduce the intermediate expression swell. We concentrate mainly in calculations over function fields and in elimination problems for which the results are very promising. Because of the flexibility of the algorithm there is a good chance to adjust it to other special problems.

The whole work is based on the principle that extra work pays off, if we can avoid extrem growth of the coefficients and the lengths of polynomials.

The algorith is presented and implemented for general ground fields and ideals as well as modules. In contrast to Faugère's F4 (where usually systems are homogenized), from which we use some ideas, it also provides a direct improvement for elimination orderings.

The algorithm described in this paper is implemented in SINGULAR 3-0-0 (still to be optimized). The command is named "slimgb".

In this paper we consider only the case of global orderings ($x_i > 1$), although it may be possible to transfer some techniques to local or mixed orderings (but e.g. the reduction algorithm would be become very complicated).

The first chapter introduces basic notations for the case of modules (the case of ideals is just a special case), the second chapter describes t-representations and an extended product criterion, the third chapter, which is the core of this article describes the slimgb-algorithm, while the last chapter presents benchmark results.

1 Classical Notions

Let $P = K[x_1, \ldots, x_n]$ be the polynomial ring over the field K. A **monomial ordering** on P, more precisely, on the set of monomials $\{x^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n} | \alpha \in \mathbb{N}^n\}$, is total ordering ">" with $x^\alpha > x^\beta \Rightarrow x^{\alpha+\gamma} > x^{\beta+\gamma}$.

We always assume that ">" is global (i.e. $x_i > 1$ for $i = 1, \ldots, n$ or, equivalently, ">" is a well ordering). We call an element $f \in P^r$ of the form $f = x^\alpha e_j$ a monomial (lying in component j), where e_j is the jth unit vector of P^r. A monomial ordering on P^r is a total ordering on the set of monomials

$$\{x^\alpha e_j | \alpha \in \mathbb{N}^n, j = 1, \ldots, r\},$$

also denoted by ">", satisfying

$$x^\alpha > x^\beta \Rightarrow x^\alpha e_i > x^\beta e_i$$

and

$$x^\alpha e_i > x^\beta e_j \Rightarrow x^{\alpha+\gamma} e_i > x^{\beta+\gamma} e_j.$$

An expression λm ($\lambda \in K$, m a monomial) is called a **term** and λ the **coefficient**. An arbitrary element $f \in P^r$ is called a **polynomial** (or a vector).

Let $f = \sum_{\alpha,i} c_{\alpha,i} \cdot \underline{x}^\alpha \cdot e_i$ ($c_{\alpha,i} \in K$) a polynomial. Then

$$\mathrm{supp}(f) := \{x^\alpha e_i | c_{\alpha,i} \neq 0\}$$

is called the **support** of f.

We say f lies completely in component j, if all monomials in the support of f ly in component j. Furthermore $\mathrm{lm}(f)$ denotes the **leading monomial** of f, the biggest monomial occuring in f w.r.t. ">" (if $f \neq 0$). The corresponding term is denoted by $\mathrm{lt}(f)$ and the coefficient by $\mathrm{lc}(f)$. Moreover, we set

$$tail(f) := f - lt(f).$$

If $F \subset P^r$ is any subset, $L(F)$ denotes the **leading ideal** of F i.e. the P-submodule of P^r generated by $\{\mathrm{lm}(f) | f \in F \backslash \{0\}\}$. The S-Polynomial of $f, g \in P^r \backslash \{0\}$ with $\mathrm{lm}(f) = x^\alpha e_i$, $lm(g) = x^\beta e_i$ is denoted by

$$\mathrm{spoly}(f, g) := x^{\gamma - \alpha} f - \frac{\mathrm{lc}(f)}{\mathrm{lc}(g)} x^{\gamma - \beta} g,$$

where $\gamma = \mathrm{lcm}(\alpha, \beta) := (\max(\alpha_1, \beta_1), \ldots, \max(\alpha_n, \beta_n))$. Note that for r=1 we get the classical notions of leading monomial, S-polynomial, etc. in the polynomial ring P itself.

Definition 1.1 (Standard representation). Let $f, g_1, \ldots g_m \in P^r$, and let $h_1, \ldots, h_m \in P$. Then

$$f = \sum_{i=1}^{m} h_i \cdot g_i \in K[x_1, \ldots, x_n],$$

is called a standard representation of f with respect to $g_1, \ldots, g_m$, if

$$\forall i : h_i \cdot g_i = 0 \text{ or otherwise } \mathrm{lm}(h_i \cdot g_i) \leq \mathrm{lm}(f)$$

The classical product criterion of Buchberger (Buchberger, 1985) for modules reads as follows:

Lemma 1.2 (Product criterion). *Let $f, g \in K[x_1, \ldots, x_n]^r$ lying entirely in the same module component (e.g. $f = F \cdot e_i$, $g = G \cdot e_i$ for F, $G \in P$. If* $\mathrm{lm}(f) \cdot \mathrm{lm}(g) = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$, *then* $\mathrm{spoly}(f, g)$ *has a standard representation w.r.t. $\{f, g\}$.*

2 T-Representations and Extended Product Criterion

There is an alternative approach to standard representations formulated in (Becker, T. and Weispfenning, V., 1993) and used in (Faugère, J.-C., 1999), which uses the notion of t-representations. While this notion is mostly equivalent to using syzygies, it makes the correctness of the algorithms easier to understand.

Definition 2.1 (t-representation). Let t be a monomial, $f, g_1, \ldots g_m \in P^r$, $h_1, \ldots, h_m \in P$. Then

$$f = \sum_{i=1}^{m} h_i \cdot g_i \in P^r$$

is called a t-representation of f with respect to $g_1, \ldots, g_m$ if

$$\forall i : lm(h_i \cdot g_i) \leq t \text{ or } h_i \cdot g_i = 0.$$

Example 2.2.

- Let the monomials of P be lexicographically ordered ($x > y$) and let
$$t = x^5y^5, g_1 = x^2, g_2 = x^5 - y, f = y$$
- Then $f = x^3g_1 - g_2$ is a x^5y^5-representation for f.
- Each standard representation of f is a lm(f)-representation.
- For $t < lm(f)$ t-representations of f do not exist.

Notation. Given a representation $p = \sum_{i=1}^{m} h_i \cdot f_i$ w.r.t. a family of polynomials $f_1, \ldots f_m$, we may shortly say that p has a nontrivial t-representation, if a t-representation of p exists with
$$t < \max\{lm(h_i \cdot f_i) | h_i \cdot f_i \neq 0\}.$$

For example $\operatorname{spoly}(f_i, f_j)$ has a nontrivial t-representation if there exists a representation of $\operatorname{spoly}(f_i, f_j)$ where the summands have leading terms smaller than
$$\operatorname{lcm}(\operatorname{lm}(f_i), \operatorname{lm}(f_j)).$$

Theorem 2.3 (extended product criterion). *Let $f = h \cdot f'$, $g = h \cdot g'$ and $f, g, f', g' \in P^r$, $h \in P$ polynomials lying completely in the same module component. If $\operatorname{lm}(f')\operatorname{lm}(g') = \operatorname{lcm}(\operatorname{lm}(f'), \operatorname{lm}(g'))$ then $\operatorname{spoly}(f, g)$ has a standard representation w.r.t. $\{f, g\}$.*

Proof. As the product criterion holds for f', g', there exist polynomials λ, μ, s.t. $\operatorname{spoly}(f', g') = \lambda f' + \mu g'$ is a standard representation. Then also $\operatorname{spoly}(f, g) = \lambda f + \mu g$ is a standard representation. □

Remark 2.4. In practise the extended product criterion will be applied only to the case where the common factor h is a monomial. In this case it is very fast to check: for each polynomial f one computes the monomial gcd m_f of all its terms. Considering the pair f, g one only has to take the gcd of the monomials m_f, m_g.

This is a reformulation of the classical Buchberger's chain criterion for t-representations.

Theorem 2.5 (t-chain criterion). *Let G be a list of polynomials, and let $f_1, f_2, \ldots, f_m \in G$ with leading monomials involving the same module component, $\operatorname{lm}(f_i) = x^{\alpha_i}e_j$, such that*
$$\forall i : x^{\alpha_i} | \operatorname{lcm}(x^{\alpha_1}, x^{\alpha_m}).$$

Assume that for all $i \in \{1, \ldots, m-1\}$, one of the two following conditions holds:

1. *$\operatorname{spoly}(f_i, f_{i+1})$ has a (nontrivial) t-representation for some monomial $t < \operatorname{lcm}(\operatorname{lm}(f_i), \operatorname{lm}(f_{i+1}))$ with respect to G.*

2. $x^{\alpha_i} \cdot x^{\alpha_{i+1}} | \operatorname{lcm}(x^{\alpha_1}, x^{\alpha_m})$ *and* f_i*,* f_{i+1} *lie completely in the same component.*

Then $\operatorname{spoly}(f_1, f_m)$ *has a (nontrivial) t-representation for some monomial* $t < \operatorname{lcm}(\operatorname{lm}(f_1), \operatorname{lm}(f_m))$ *with respect to G.*

This is a generalization of results in (Becker, T. and Weispfenning, V., 1993), the trivial syzygy approach is inspired by (Faugère, J.-C., 2002).

Proof. Without loss of generality we may assume that all f_i are normalized. Then

$$\begin{aligned} &\operatorname{spoly}(f_1, f_m) \\ &\quad = \quad t_1 \cdot \operatorname{spoly}(f_1, f_2) + t_2 \cdot \operatorname{spoly}(f_2, f_3) + \ldots + t_{m-1} \cdot \operatorname{spoly}(f_{m-1}, f_m) \end{aligned}$$

where

$$t_i = \operatorname{lcm}(\operatorname{lm}(f_1), \operatorname{lm}(f_m)) / \operatorname{lcm}(\operatorname{lm}(f_i), \operatorname{lm}(f_{i+1})).$$

Therefore, it remains to show, that each summand has nontrivial t-representation (precisely some $t < \operatorname{lcm}(\operatorname{lm}(f_1), \operatorname{lm}(f_m))$).

If condition 1 holds, then the multiplication by the monomial t_i gives just a good t, which satisfies our condition. In the second case one has the trivial syzygy (using $f_i = p_i \cdot e_j$, $f_{i+1} = p_{i+1} \cdot e_j$) $p_i \cdot f_{i+1} - p_{i+1} \cdot f_i = 0$. Therefore

$$\operatorname{lm}(p_{i+1}) \cdot f_i - \operatorname{lm}(p_i) \cdot f_{i+1} = \operatorname{tail}(p_i) \cdot f_{i+1} - \operatorname{tail}(p_{i+1}) \cdot f_i.$$

Since condition (2) holds, we have only to multiply both sides with

$$\operatorname{lcm}(x^{\alpha_i}, x^{\alpha_m}) / (x^{\alpha_i} \cdot x^{\alpha_{i+1}}).$$

□

3 The Slimgb Algorithm

In this section we describe the new algorithm slimgb. If F is a finite tuple $F[i]$ denotes the ith entry of F.

3.1 The Main Algorithm

The main algorithm is a mixture of the classical Buchberger and Faugère's F4 (see Algorithm 1).

Remark 3.1. When choosing critical pairs from P, one should of course take only pairs with minimal sugar value (see Giovini et al. (1991)). One is free to use any kind of criteria to omit useless pairs, as long as the criteria are compatible, i.e. the question wether a pair is critical depends only on the tuple $(\operatorname{lm}(F[1]), \ldots, \operatorname{lm}(F[\#F]))$.

For example, the chain criterion and the classical product criterion are compatible. Also Hilbert driven Buchberger techniques belong to this category and are compatible.

Algorithm 1 slimgb main procedure, calculates a Gröbner basis of F

Require: F finite tuple of polynomials (from $K[x_1, \ldots, x_n]^r$).
  $P := \{(i, j)|1 \le i < j \le \#F$ and $\mathrm{lm}(F[i]), \mathrm{lm}(F[j])$ ly
  the same component$\}$ /* P is the set of critical pair indices */
  apply criteria to P
  while $P \neq \emptyset$ **do**
    choose $\emptyset \neq S \subset P$
    $P := P \backslash S$
    $(R, F) :=$ slimgbReduce(S, F) /* where slimgbReduce is specified later */
    for $0 \neq r \in R$ **do**
      $F :=$ append(F, r)
      $P := P \cup \{(i, \#F)|1 \le i < \#F\}$
      apply criteria to P
    end for
  end while
  return F

In particular the approach in this paper is completely orthogonal to the major advancements done in (Caboara, M.; Kreuzer M.; Robbiano, L., 2004). So a combination of these techniques is recommended.

The extended product criterion is also admitted, as it sorts out pairs, which have, for a temporary state of the list F, nontrivial t-representation. By the following specification of the reduction algorithm this property is never lost.

3.2 The Reduction Algorithm Specification

The reduction algorithm slimgbReduce takes as arguments the index set S (subset of the pair set), and a finite tuple of polynomials F (the generating system) and gives back a new set R and a finite tuple of polynomials F', s.t. the following holds:

1. $\langle F' \cup R \rangle_{K[x_1,\ldots,x_n]} = \langle F \rangle_{K[x_1,\ldots,x_n]}$,
2. $\#F = \#F'$,
3. F' preserves the order of F: $\forall i : \mathrm{lm}(F[i]) = \mathrm{lm}(F'[i])$ and $\mathrm{spoly}(F[i], F'[i])$ has standard representation w. r. t. $F' \cup R$,
4. each $r \in R$ has a nontrivial t-representation w. r. t. $F' \cup R$,
5. and for termination: $R \neq \emptyset \Rightarrow \exists r \in R : lm(r) \notin L(F)$.

Remark 3.2. The third condition implies, that each $f \in F$ has standard representation w.r.t. $F' \cup R$. This implies again, that for each (i, j) the fact that $\mathrm{spoly}(F[i], F[j])$ has nontrivial t-representation with respect to F implies that the index pair (i, j) has the same property with respect to $F' \cup R$. The application of the classical Buchberger normal form of (Greuel, G.-M. and Pfister, G.,

2002) for all elements in R fullfills the desired properties of a reduction algorithm for slimgb. Note also that using Gaussian elimination like in Faugère's F4 is a valid reduction algorithm. We will introduce a different algorithm featuring parallel reduction, modifying F by following our main strategy.

3.3 Termination and Correctness of the Main Algorithm

Theorem 3.3. *Let $f_1, \ldots, f_k \in K[x_1, \ldots, x_n]^r$, and let $F = (f_1, \ldots, f_k)$. For i, j with $\mathrm{lm}(f_i), \mathrm{lm}(f_j)$ lying in the same module component define $m_{i,j}$ as follows: If $\mathrm{lm}(f_i) = x^\alpha \cdot e_l$, $\mathrm{lm}(f_j) = x^\beta \cdot e_l$, then define $m_{i,j} = \mathrm{lcm}(x^\alpha, x^\beta)/x^\alpha$. The following are equivalent:*

1. *F is a Gröbner basis of $\langle F\rangle_{K[x_1,\ldots,x_n]}$*
2. *Set*

$$M := \left\{ (i,j) \;\middle|\; \begin{array}{l} 1 \le i < j \le \#F, \\ \mathrm{lm}(F[i]), \mathrm{lm}(F[j]) \textit{ are in the same component} \end{array} \right\}.$$

Then, for any subset $J \subset M$ with

$$\begin{aligned} \langle \{m_{i,j} \cdot e_i | (i,j) \in J\}\rangle_{K[x_1,\ldots,x_n]} &= \langle \{m_{i,j} \cdot e_i | (i,j) \in M\}\rangle_{K[x_1,\ldots,x_n]} \\ &\subseteq K[x_1, \ldots, x_n]^r \end{aligned}$$

$\mathrm{spoly}(f_i, f_j)$ *has a nontrivial t-representation w.r.t. F (i.e. for some $t < \mathrm{lcm}(\mathrm{lm}(f_i), \mathrm{lm}(f_j))$) $\forall (i,j) \in J$.*

This is a simple generalization of results from (Greuel, G.-M. and Pfister, G., 2002).

Proof. The proof is analogous to the proof of the Schreyer theorem in (Greuel, G.-M. and Pfister, G., 2002, p. 142). □

Theorem 3.4 (Termination and correctness). *Algorithm 1 terminates and is correct.*

Proof. In each loop of the iteration either the reduced R is empty, then the set of critical pairs becomes smaller, or there is a f in R with $f \notin \mathrm{L}(F)$. Hence the algorithm terminates, if the reduction algorithm is correct and terminates.

By the properties of the reduction algorithm, $\langle F\rangle$ remains constant. Each critical index pair has a nontrivial t-representation at a certain point. By remark 3.2 this property is preserved.

Concerning compatibility with changing the elements of F note that $\mathrm{lm}(F[i])$ is constant at every time. If $\mathrm{spoly}(F[i], F[j])$ has a nontrivial t-representation at some point, this property will be preserved in the future. So together with our compatibility condition for the criteria it follows that every critical pair has nontrivial t-representation in the end. Therefore, the set F is a Gröbner basis of the module generated by the original F (the input), when the algorithm terminates. This finishes the proof of correctness for the algorithm. □

3.4 The Strategy Component: Weighted Lengths

One key feature of slimgb is the flexible, problem oriented choice of a good weighted length algorithm. By specifying a weighted length, the algorithm gains a good criterion, which choices during the algorithm (e.g. which polynomial to use for reduction, exchange elements of F by better ones, selection of critical pairs, pivoting) lead to intermediate expression swell and which prevent them.

We introduce now the notion of a weighted length by requiring certain properties which seem to be reasonable.

Definition 3.5 (weighted length). A weighted length function is a map $\mathrm{wlen} : K[x_1, \ldots, x_n]^r \to \mathbb{N}$ with the following properties:

- $\mathrm{wlen}(p) = 0 \Leftrightarrow p = 0$
- $\mathrm{wlen}(p) \geq \#\,\mathrm{supp}(p)$
- if t is a term, $\mathrm{lm}(t) \notin \mathrm{supp}(p)$, and $\mathrm{lm}(p) > lm(t)$ then $\mathrm{wlen}(p + t) > \mathrm{wlen}(p)$
- m a monomial $\Rightarrow \mathrm{wlen}(p) = \mathrm{wlen}(m \cdot p)$

However none of these properties is required for the algorithm to be correct, but they are essential for the performance. Maybe in some settings one has to drop certain properties. For example, when extending this algorithm to the noncommutative case, the wlen might be nonconstant under multiplication with a monomial (as this is already the case with the normal length).

Important weighted length functions are:

- length: $\mathrm{wlen}(p) = \#\,\mathrm{supp}(p)$
- coefficient strategy length:

  $$\mathrm{wlen}(p) = \mathrm{coeffSize}(\mathrm{lc}(p)) \cdot \#\,\mathrm{supp}(p)$$

  (works very good over function fields)
- elimination strategy length:

  $$\mathrm{wlen}(p) = \sum_{m \in \mathrm{supp}(p)} (1 + |\deg(m) - \deg(\mathrm{lm}(p))|^{+})$$

  (good for elimination orderings, $|x|^{+} := x$ if x is positive, 0 else).
- elimination and coefficient strategy length:

  $$\mathrm{wlen}(p) = \mathrm{coeffSize}(\mathrm{lc}(p)) \cdot \sum_{m \in \mathrm{supp}(p)} (1 + |\deg(m) - \deg(\mathrm{lm}(p))|^{+})$$

Remark 3.6. When using buckets as polynomial data structure, the length can only be estimated (but this is no serious problem). The elimination strategy weighted length uses the algebraic structure, it cannot easily be transfered to linear algebra like in Faugère's F4 (Faugère, J.-C., 1999). The elimination strategy speeds up many lexicographical examples up to a factor of 1000 (compared to the same algorithm with normal lengths).

3.5 The Reduction Algorithm of Slimgb

The idea of the reduction algorithm is to eliminate effects of disadvantageous order of computations. For this aim the reduction is done parallel, and generating system members of the considered system F are exchanged by better ones, which are found later.

The reduction algorithm works similar to Gaussian elimination with extra operations.

First we assign

$$R := \{\mathrm{spoly}(F[i], F[j]) | (i, j) \in S\}$$

We say that an operation takes place at a monomial m, if m is the highest leading monomial of the polynomials that is involved in the operation. We define:

$$R_m := \{r \in R | \operatorname{lm}(r) = m\}$$

We use an operation on the system R, which takes place at the highest leading monomial m, for which such an operation is possible (analogous to Gaussian elimination). After each operation, we filter out zero entries from R.

This is iterated as long as possible.

The allowed operations are:

1. $f \in F, \operatorname{lm}(f)$ divides m, then $R := R \backslash R_m \cup \{\mathrm{spoly}(r, f) | r \in R_m\}$
2. $r \in R, \#R_m \geq 2$, then $R := R \backslash R_m \cup \{r\} \cup \{\mathrm{spoly}(r, f) | f \in R_m \backslash \{r\}\}$
3. $r \in R,\ f = F[i],\ \operatorname{lm}(f) = \operatorname{lm}(r),\ \mathrm{wlen}(f) > \mathrm{wlen}(r),$ then $F[i] := r$ and $R := R \backslash \{r\} \cup \{\mathrm{spoly}(r, f)\}$

At the end the new F and R are returned.

Remark 3.7. There is sometimes a choice, whether to use the first or the second operation. This is decided by the "best" (i.e. with the lowest weighted lengths) polynomial one finds with leading monomial dividing m. If it is contained if F, the first operation is used, otherwise the second.

Proof. Termination is clear, since R is finite and the leading monomials drop (well ordering).

Correctness: Note that all properties of a valid reduction algorithm (exception: the termination property) hold at the beginnig of the reduction algorithm and in each step this property is preserved. If no operation is possible any more, then either R is empty, or R is nonempty and operation (1) is not possible any more. Hence there exist $r \in R$ s.t. $\text{lm}(r) \notin \text{L}(F)$. $\square$

4 Notes on the Implementation of the Algorithm

- Even if the reduction algorithm is introduced by using sets, they shouldn't be used in the implementation, i.e. the "slimgb" algorithm in SINGULAR uses ordered list (which are limited to a size of 100, as the sorting routines are too expensive on big list).

- Generalizing the second operation in the reduction algorithm by reducing some element f by an element r, if $\text{lm}(r)$ divides $\text{lm}(f)$ (instead of equality), appears not to be a good idea. In elimination examples the performance drops by factor 1000 and more. One plausible explanation is: Assuming the sugar value of f is equal to the sugar value of r, then the sugar value of $\text{spoly}(f, r)$ is worse than the sugar value of f (if $\text{lm}(r)$ only divides $\text{lm}(f)$).

- Although no data structures are prescribed, geobuckets are recommended during the reduction algorithm for the elements of R. The elements stay in the form of geobuckets until they are used to reduce something (operation 2). Then they are brought into a canonical form, used for reduction and converted back to geobuckets. For calculating the length of a polynomial the bucket structure should not be dissolved. Either one estimates the length during generation by the formula $\text{length}(\text{spoly}(f, g)) \leq \text{length}(f) + \text{length}(g) - 2$ or one addd up the lengths of the buckets.

- If the coefficient space is $\mathbb{Q}$ or some function field, denominators should be avoided. In particular, in our experiments we assume that all coefficients lie in $\mathbb{Z}$ or in the polynomial ring in case of function fields. These are only rings and therefore content plays a role. Whenever adding an element to F or using an element from R to reduce others, the content should be extracted.

5 Benchmarks

5.1 The Algorithms

This section compares "slimgb" and "std". While "slimgb" is the implementation in SINGULAR of the algorithm presented in this paper, "std" is the standard Buchberger algorithm in Singular, which is highly optimized. As "slimgb" and "std" share both the same data structures and many other functions, they are comparable w.r.t. the quality of the implementation, even if "std" is to some degree more optimized.

It is clear that our algorithm has the same worst-case complexity as Buchberger's algorithm. Therefore we can only report on practical efficiency by comparing implementations based on the the same environment, in particular on the same data structures. Moreover, as there is no reduction strategy, which is uniformly preferred on all examples[1], we present a variety of different examples, all chosen from Symbolic Data (SymbolicData (2000-2002)) such that everyone can recompute them. Although not uniformly good, our reduction strategy seems to be generally better when extrem expression swell occurs during the computation (e.g. computations over function fields or elimination ordering) . "slimgb" uses at the moment weighted length for coefficient strategy over function fields and over the rationals in the case of homegenous systems or degree orderings . If the field is $\mathbb{Z}/p$ and the ordering is a elimination ordering an elimination strategy is used. If the field is $\mathbb{Z}/p$ and the example is homogeneous or has a degree ordering the usual length is used. Over function fields, if the ordering is an elimination ordering, then the combined weighted length for elimination and coefficients is used.

5.2 The Results

In the following some benchmarks are presented.

The used Singular version is compiled from the sources file "Singular-3-0-0-4.tar.gz" (with one additional bugfix), freely downloadable.

The machine used for these benchmarks is a Dual Opteron with 2,2 Ghz and 16 GB RAM.

Some examples took too long in std or consumed to much memory. In particularly Katsura 6 and some zerodimensional examples in lexicographic ordering were aborted manually, when using to much RAM (at least 10 GB).

The polynomial rings are denoted by $K(n)[m]$, where n is the number of parameters and m the number of variables. The field $\mathbb{F}$ stands for $\mathbb{Z}/32003$. The tables present computation time and memory consumption of the computations. The abbreviation eq. stands for the number of equations of the example.

The first table presents examples in lexicographical ordering.

[1]Despite this fact, some systems claim priority over others by selecting a few very special examples and often, when computing other examples, the picture is very different.

example	ring	eq.	slimgb		std	
Katsura 6	$\mathbb{F}[7]$	7	$0.19s$	$2.5MB$	$> 1819s$	$>21000MB$
Katsura 5	$\mathbb{F}[6]$	6	$0.01s$	$0.8MB$	$1.80s$	$69.8MB$
G./Chou 303 1	$\mathbb{F}(5)[8]$	8	$2.60s$	$6.3MB$	$>158370s$	$> 2200MB$
G./Chou 267 1	$\mathbb{F}(4)[3]$	3	$0.05s$	$0.8MB$	$0.05s$	$0.8MB$
G./Morley 1	$\mathbb{F}(2)[6]$	6	$4.77s$	$6.3MB$	$71.96s$	$29.8MB$
G./Chou 274 2	$\mathbb{F}(4)[7]$	7	$102s$	$26.8MB$	$>163709s$	$> 500MB$
G./Chou 238 1	$\mathbb{F}(3)[8]$	8	$0.01s$	$0.8MB$	$0.03s$	$0.8MB$
G./Chou 302 1	$\mathbb{F}(5)[8]$	8	$2.32s$	$9.8MB$	$>150634s$	$> 2700MB$
G./Brahmagupta	$\mathbb{F}(4)[5]$	5	$0.18s$	$1.3MB$	$883s$	$61.3MB$
ZeroDim/example 26	$\mathbb{F}[2]$	2	$0.01s$	$0.8MB$	$0.08s$	$6.8MB$
ZeroDim/example 23	$\mathbb{F}[6]$	6	$1.54s$	$1.3MB$	$0.65s$	$2.3MB$
ZeroDim/example 62	$\mathbb{F}[3]$	3	$0.03s$	$0.8MB$	$0.02s$	$1.3MB$
ZeroDim/example 33	$\mathbb{F}[7]$	7	$0.25s$	$0.9MB$	$0.10s$	$1.3MB$
ZeroDim/example 57	$\mathbb{F}[8]$	8	$0.30s$	$3.0MB$	$> 1591s$	$>15000MB$
ZeroDim/example 16	$\mathbb{F}[3]$	3	$0.05s$	$1.3MB$	$0.47s$	$28.3MB$
ZeroDim/example 54	$\mathbb{F}[6]$	6	$3.39s$	$1.3MB$	$0.81s$	$1.8MB$
ZeroDim/example 51	$\mathbb{F}[3]$	3	$0.07s$	$1.3MB$	$0.02s$	$1.8MB$
ZeroDim/example 7	$\mathbb{F}[3]$	3	$0.01s$	$0.8MB$	$0.11s$	$9.8MB$
ZeroDim/example 46	$\mathbb{F}[4]$	4	$0.07s$	$0.8MB$	$0.02s$	$1.3MB$
ZeroDim/example 58	$\mathbb{F}[4]$	4	$0.05s$	$0.8MB$	$0.04s$	$2.3MB$
ZeroDim/example 29	$\mathbb{F}[8]$	8	$0.03s$	$0.8MB$	$> 1451s$	$>15000MB$

The second table contains examples in degree reverse lexicographical ordering.

example	ring	eq.	slimgb		std	
H./G./Biarc	$\mathbb{Q}(3)[6]$	4	$0.01s$	$0.6MB$	$0.07s$	$1.1MB$
H./G./Simson 3	$\mathbb{F}(4)[10]$	9	$128s$	$128MB$	$> 66735s$	$> 13160MB$
H./G./Butterfly 1	$\mathbb{Q}(4)[8]$	7	$0.86s$	$6.1MB$	$110s$	$103MB$
H./G./Brahmagupta	$\mathbb{Q}(4)[6]$	5	$0.02s$	$0.6MB$	$0.09s$	$0.6MB$
G./Chou 238 1	$\mathbb{Q}(3)[8]$	8	$2.32s$	$1.1MB$	$2.08s$	$1.1MB$
G./Chou 302 1	$\mathbb{Q}(5)[8]$	8	$1.46s$	$3.6MB$	$4.01s$	$6.1MB$
G./Chou 106 1	$\mathbb{Q}(4)[6]$	6	$40.92s$	$8.1MB$	$100s$	$32.6MB$
G./Chou 267 1	$\mathbb{Q}(4)[3]$	3	$0.04s$	$0.6MB$	$0.02s$	$0.6MB$
G./Chou 249 2	$\mathbb{Q}(3)[5]$	5	$0.01s$	$0.6MB$	$0.05s$	$0.6MB$
G./Chou 303 1	$\mathbb{Q}(5)[8]$	8	$5.21s$	$7.1MB$	$15.60s$	$17.1MB$
G./Chou 290 1	$\mathbb{Q}(4)[7]$	7	$0.00s$	$0.6MB$	$0.06s$	$1.1MB$
G./Chou 106 2	$\mathbb{Q}(4)[6]$	6	$0.03s$	$0.6MB$	$0.03s$	$0.6MB$
G./Chou 304 1	$\mathbb{Q}(4)[10]$	10	$1.51s$	$1.1MB$	$1.92s$	$1.1MB$
G./Chou 249 3	$\mathbb{Q}(3)[8]$	8	$0.01s$	$0.6MB$	$0.05s$	$0.6MB$
G./Chou 296 2	$\mathbb{Q}(4)[9]$	9	$0.02s$	$0.6MB$	$0.71s$	$0.6MB$
G./Butterfly 1	$\mathbb{Q}(4)[7]$	7	$1.40s$	$7.1MB$	$14.25s$	$31.1MB$
G./Chou 131 1	$\mathbb{Q}(3)[6]$	6	$0.02s$	$0.6MB$	$0.03s$	$0.6MB$
G./Chou 201 1	$\mathbb{Q}(3)[6]$	6	$0.01s$	$0.6MB$	$0.04s$	$0.6MB$
G./Chou 274 2	$\mathbb{Q}(4)[7]$	7	$0.39s$	$1.1MB$	$1184s$	$86.1MB$
G./Chou 160 1	$\mathbb{Q}(3)[6]$	6	$0.10s$	$0.6MB$	$16.23s$	$8.6MB$

example	ring	eq.	slimgb		std	
G./Chou 167 1	$\mathbb{Q}(3)[8]$	8	$9.50s$	$1.1MB$	$9.24s$	$1.6MB$
G./Chou 116 1	$\mathbb{Q}(3)[3]$	3	$0.02s$	$0.6MB$	$0.05s$	$0.6MB$
G./Chou 156 1	$\mathbb{Q}(3)[4]$	4	$0.03s$	$0.6MB$	$0.06s$	$0.6MB$
G./Chou 128 1	$\mathbb{Q}(3)[4]$	4	$0.03s$	$0.6MB$	$0.30s$	$0.6MB$
G./Chou 115 2	$\mathbb{Q}(3)[8]$	8	$432s$	$199MB$	$2662s$	$88.6MB$

The timings show that in the cases of function fields and elimination problems "slimgb" is by far better than std.

Of course, one can compare slimgb also to some indirect algorithms. You can compare of "slimgb" to the old version of "groebner" (without "slimgb" as backend for function fields), which first homogenizes, calculates some degree reverse lexicographical Gröbner basis, then uses the Hilbert driven Buchberger algorithm for a lexicographical Gröbner basis and finally dehomogenizes. The for half of elimination examples over normal fields "slimgb" is by far better, on half "groebner" is by far better. The differences are in both directions huge. However, these are completely different algorithms and we need them both. Therefore it is important that with slimgb the direct algorithms regain ground.

In elimination examples over function fields "slimgb" is much better than "groebner".

Remark 5.1. Possible improvements include:

- find some good weighted length for characteristic 0 examples
- a noncommutative version of "slimgb" (wil be part of Singular 3-0-1, results seem to be even more promising than in the commutative case, even the coefficient growth of rational numbers, which is generated by commutation relations, is kept under control)
- optimize the implementation of the slimgb
- try some linear algebra backends for the reduction
- implement an option to let the user select the strategy
- use "slimgb" as backend for some indirect (e. g. Hilbert driven) approach for calculating Gröbner bases.

Acknowledgements

I'd like to thank my supervisor G.-M. Greuel for his support and the big amount of time he always took for me, I also thank H. Schönemann for sharing his knowledge about the implementation of Gröbner bases and his help with the implementation questions. Furthermore I want to thank all the colleagues, who helped me to improve this article.

References

Becker, T. and Weispfenning, V., 1993. Gröbner bases, a computational Approach to Commutative Algebra. Graduate Texts in Mathematics, Springer Verlag.

Buchberger, B., 1985. A Criterion for Detecting Unnecessary Reductions in the Construction of a Gröbner Basis. In: Bose, N. K. (Ed.), Recent trends in multidimensional system theory.

Buchberger, B., 1965. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal. Dissertation, Universitt Insbruck.

Caboara, M.; Kreuzer M.; Robbiano, L., 2004. Efficiently Computing Minimal Sets of Critical Pairs. Journal of Symbolic Computation, 38, 1169–1190.

Faugère, J.-C., 1999. A new Efficient Algorithm for Computing Gröbner Bases (F_4). Journal of Pure and Applied Algebra 139 (1–3), 61–88.

Faugère, J.-C., 2002. A new Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F_5). In: Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'02). ACM Press, pp. 75–83.

Giovini, A., Mora, T., Niesi, G., Robbiano, L., Traverso, C., 1991. One sugar cube, please or Selection strategies in Buchberger algorithms. In: Watt, S. (Ed.), Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computations, ISSAC'91. ACM press, pp. 49–54.

Greuel, G.-M., Pfister, G., Schönemann, H., 2005. SINGULAR 2.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern, `http://www.singular.uni-kl.de`.

Greuel, G.-M. and Pfister, G., 2002. A SINGULAR Introduction to Commutative Algebra. Springer Verlag.

SymbolicData, 2000-2002. The Symbolic Data Project. `http://www.SymbolicData.org`

Yan, T., March 1998. The geobucket data structure for polynomials. J. Symbolic Computation 25 (3), 285–294.

Papers published in the Reports on Computer Algebra series

[RCA:34] E.A. Tobis. Libraries for Counting Real Roots. July 2005.

[RCA:33] A. Frühbis-Krüger and G. Pfister. Practical Aspects of Algorithmic Resolution of Singularities. October 2004.

[RCA:32] A. Frühbis-Krüger, K. Krüger, and H. Schönemann. Dynamic Modules in SINGULAR. December 2003.

[RCA:31] G.-M. Greuel, C. Lossen, and M. Schulze. Three Algorithms in Algebraic Geometry, Coding Theory, and Singularity Theory. March 2001.

[RCA:30] G.-M. Greuel. Applications of Computer Algebra to Algebraic Geometry, Singularity Theory and Symbolic-Numerical Solving. June 2000.

[RCA:29] G.-M. Greuel. Computer Algebra and Algebraic Geometry – Achievements and Perspectives. Feb 2000.

[RCA:28] T. Siebert. Recursive Computation of Free Resolutions and a Generalized Koszul Complex. Jan 2000.

[RCA:27] O. Bachmann and H.-G. Gräbe. The SymbolicData Project: Towards an Electronic Repository of Tools and Data for Benchmarks of Computer Algebra Software. Jan 2000.

[RCA:26] B. Reinert. Solving One-Sided Equations in Integer Monoid Rings. Jan 2000.

[RCA:25] B. Reinert and D. Zeckzer. Coset Enumeration using Prefix Gröbner Bases in Mrc – An Experimental Approach. Nov 1999.

[RCA:23] Birgit Reinert. Observations on coset enumeration. Nov 1998.

[RCA:22] K. Madlener and F. Otto. Some Applications Of Prefix-Rewriting In Monoids, Groups, And Rings. November 1998.

[RCA:21] G.-M. Greuel, G. Pfister, and H. Schönemann. Singular version 1.2 user manual. June 1998.

[RCA:20] B. Reinert and D. Zeckzer. MRC – A System for Computing Gröbner Bases in Monoid and Group Rings. July 1998.

[RCA:19] B. Reinert, K. Madlener, and T. Mora. A note on nielsen reduction and coset enumeration. February 1998.

[RCA:18] O. Bachmann and H. Schönemann. Monomial Representations for Gröbner Bases Computations. January 1998.

[RCA:17] Thomas Siebert. An algorithm for constructing isomorphisms of modules. January 1998.