Improving incremental signature-based Gröbner basis algorithms

Christian Eder
c/o Department of Mathematics
TU Kaiserslautern
67653 Kaiserslautern, Germany
ederc@mathematik.uni-kl.de

Abstract

In this paper we describe a combination of ideas to improve incremental signature-based Gröbner basis algorithms having a big impact on their performance. Besides explaining how to combine already known optimizations to achieve more efficient algorithms, we show how to improve them even more. Although our idea has a psotive affect on all kinds of incremental signature-based algorithms, the way this impact is achieved can be quite different. Based on the two best-known algorithms in this area, F5 and G2V, we explain our idea, both from a theoretical and a practical point of view.

1 Introduction

Computing Gröbner bases is a fundamental tool in commutative and computer algebra. Buchberger introduced the first algorithm to compute such bases in 1965, see [2]. In the meantime lots of additional and improved algorithms have been developed.

In the last couple of years, so-called *signature-based* algorithms like F5, see [7], and G2V, see [8], have become more popular. Lots of optimizations for these algorithms have been published, for example, see [1, 5, 9, 14]. Whereas the last two publications focus on the field of non-incremental signature-based algorithms, we focus our discussion in this paper to the initial presentation of this kind of algorithms, based on Faugère's initial presentation of F5 in [7]: Computing Gröbner bases step by step iterating over the generators of the input system. The intermediate states of this incremental structure can be used to improve performance.

The intention of this paper is not only to cover, to collect, and to compare the various optimizations found recently, but also to increase the algorithms' efficiency. As discussed in-depth in [6], signature-based algorithms differ mainly on their implementation of two criteria used to detect useless critical pairs during the computations, the *non-minimal signature* criterion and the *rewriting signature* criterion; the optimizations presented in this publication have mostly an impact on the first one criterion. We focus our discussion on the two best-known and most efficient incremental algorithms in this area, namely F5 and G2V. Due to their different, in some sense even opposed, usages of the above mentioned criteria, their behaviour under our optimizations gives a rather accurate picture of the general impact on the class of incremental signature-based algorithms.

In Section 2 we introduce the basic notions of incremental signature-based algorithms. In [5] the idea of interreducing intermediate Gröbner bases between the iteration steps of F5 is illustrated: Speed-ups of up to 30% comparing F5C to the basic F5 can be achieved by minimizing the computational overhead generated due to the inner workings of signature-based algorithms. Section 3 shortly reviews this idea, from a more general point of view than it was done in its initial presentation back then, taking its effects on algorithms like G2V into account. G2V and the idea of using zero reductions actively in the current iteration step is content of Section 4. The idea of using recent zero reductions in the algorithm goes back to Alberto Arri's preprint of [1] in 2009, where such a kind of optimization was mentioned for the first time. Combining these two, at a first look quite separated improvements in a clever way is the main contribution of this paper: We see that a quite easy idea can be used to get a faster detection of useless critical pairs; in the situation of G2V one even discards more elements, which leads to a huge improvement in the overall performance of the algorithm.

2 Basic setting

We start with some basic notations. Let $i \in \mathbb{N}$, $\mathcal{K}$ a field, and $\mathcal{R} = \mathcal{K}[x_1, \ldots, x_n]$. Let $F_i = (f_1, \ldots, f_i)$, where each $f_j \in \mathcal{R}$, and $I_i = \langle F_i \rangle \subset \mathcal{R}$ is the ideal generated by the elements of F_i. Moreover, we fix a *degree-compatible ordering* $<$ on the monoid $\mathcal{M}$ of monomials of $x_1, \ldots, x_n$. For a polynomial $p \in \mathcal{R}$, we denote p's *leading monomial* by $\operatorname{lm}(p)$, its *leading coefficient* by $\operatorname{lc}(p)$, and write $\operatorname{lt}(p) = \operatorname{lc}(p) \operatorname{lm}(p)$ for its *leading term*. For any two polynomials $p, q \in \mathcal{R}$ we use the shorthand notation

$$\tau(p, q) = \operatorname{lcm}(\operatorname{lm}(p), \operatorname{lm}(q))$$

for the *least common multiple* of their leading monomials.

Let $e_1, \ldots, e_m$ be the canonical generators of the free $\mathcal{R}$-module $\mathcal{R}^m$. We extend the ordering $<$ to a well-ordering $\prec$ on the set $\{te_j \mid t \in \mathcal{M}, 1 \leq j \leq m\}$ in the following way[1]: $t_j e_j \prec t_k e_k$ iff $j < k$, or $j = k$ and $t_j < t_k$. We define maps

$$\begin{array}{cccc} \pi : & \mathcal{R}^i & \to & I_i \\ & \sum_{j=1}^{i} p_j e_j & \mapsto & \sum_{j=1}^{i} p_j f_j, \end{array}$$

where p_j is a polynomial in $\mathcal{R}$ for $1 \leq j \leq i$. An element $\omega \in \mathcal{R}^i$ with $\pi(\omega) = 0$ is called a *syzygy of* $f_1, \ldots, f_i$. The module of all such syzygies is denoted $\operatorname{Syz}(F_i)$. A syzygy of type $f_k e_j - f_j e_k$ is called a *principal syzygy*. The submodule of all principal syzygies of F_i is denoted $\operatorname{PSyz}(F_i) \subseteq \operatorname{Syz}(F_i)$. Note that if a sequence F_i of polynomials is regular, then $\operatorname{PSyz}(F_i) = \operatorname{Syz}(F_i)$.

Let $f_{i+1} \in \mathcal{R} \backslash I_i$. We describe algorithms that, given a Gröbner basis G_i of I_i, computes a Gröbner basis of $I_{i+1} = \langle F_{i+1} \rangle$, where $F_{i+1} = (f_1, \ldots, f_{i+1})$. Thus we restrict ourselves to an incremental approach in this paper.

In [6] the class of signature-based Gröbner basis algorithms is introduced. Those give a new point of view on the computations taking so-called *signatures* into account.

Definition 2.1. Let $p \in I_{i+1}$, $j \in \mathbb{N}$ with $j \leq i + 1$, and $h_1, \ldots, h_j \in \mathcal{R}$ such that $h_j \neq 0$ and

$$p = h_1 f_1 + \cdots h_j f_j.$$

1. If $t = \operatorname{lm}(h_j)$, we say that te_j is *a signature* of p. Let $\mathcal{S}$ be the set of all signatures:
$$\mathcal{S} = \{te_j \mid t \in \mathcal{M}, \ j = 1, \ldots, i + 1\}.$$
2. Using the well-ordering $\prec$ on $\mathcal{S}$ we can identify for each $p \in \mathcal{R}$ a unique, minimal signature.
3. For any monomial $u \in \mathcal{M}$ and any signature $te_j \in \mathcal{S}$ we define a multiplication $u \cdot te_j := (ut)e_j$.
4. An element $f = (te_j, p) \in \mathcal{S} \times I_{i+1}$ is called a *labeled polynomial*. For a labeled polynomial $f = (te_j, p)$ we define the shorthand notations $\operatorname{poly}(f) = p$, $\operatorname{sig}(f) = te_j$, and $\operatorname{index}(f) = j$. Talking about the leading monomial, leading term, and leading coefficient of a labeled polynomial f we always assume the corresponding value of $\operatorname{poly}(f)$. In the same sense we define the least common multiple of two labeled polynomials f and g, $\tau(f, g)$, by $\tau(\operatorname{poly}(f), \operatorname{poly}(g))$. Furthermore, if $G = \{g_1, \ldots, g_\ell\} \subset \mathcal{S} \times I_{i+1}$, then we define
$$\operatorname{poly}(G) := \{\operatorname{poly}(g_1), \ldots, \operatorname{poly}(g_\ell)\} \subset I_{i+1}.$$
5. A *critical pair of two labeled polynomials f and g* is a tuple $(f, g) \in (\mathcal{S} \times I_{i+1})^2$.
6. Moreover, we define the *s-polynomial of two labeled polynomials f and g* by
$$\operatorname{spoly}(f, g) = \left(\omega, u_f \cdot \operatorname{poly}(f) - \frac{\operatorname{lc}(f)}{\operatorname{lc}(g)} u_g \cdot \operatorname{poly}(g)\right)$$
where
   (a) $u_f = \frac{\tau(f,g)}{\operatorname{lm}(f)}$, $u_g = \frac{\tau(f,g)}{\operatorname{lm}(g)} \in \mathcal{M}$, and

[1] Note that this differs slightly from the ordering given in [7], but our discussion is mainly based on [6]. Moreover, it simplifies notation.

(b) $\omega = \max\{u_f \mathrm{sig}(f), u_g \mathrm{sig}(g)\}$.

Adopting the notions of reduction and standard representation from the pure polynomial setting we get:

Definition 2.2. Let $f, g \in \mathcal{S} \times I_{i+1}$ be labeled polynomials, and let $G \subset \mathcal{S} \times I_{i+1}$, with $\#G = \ell$.

1. f *reduces sig-safe to* g *modulo* G if there exist sequences $j_1, \ldots, j_\ell \in \mathbb{N}$, $t_1, \ldots, t_\ell \in \mathcal{M}$, $c_1, \ldots, c_\ell \in \mathcal{K}$, and $r_0, \ldots, r_\ell \in \mathcal{S} \times I_{i+1}$ such that for all $i \in \{1, \ldots, \ell\}$ $g_{j_i} \in G$,
   (a) $r_0 = f$, $r_i = r_{i-1} - c_i t_i g_{j_i}$, $r_l = g$,
   (b) $\mathrm{lm}(r_i) < \mathrm{lm}(r_{i-1})$, and
   (c) $t_i \mathrm{sig}(g_{j_i}) \prec \mathrm{sig}(r_{i-1})$.
2. We say that f has a *standard representation with respect to* G if there exist $h_1, \ldots, h_\ell \in \mathcal{R}$, $g_1, \ldots, g_\ell \in G$ such that
   (a) $\mathrm{poly}(f) = h_1 \,\mathrm{poly}(g_1) + \cdots + h_\ell \,\mathrm{poly}(g_\ell)$,
   (b) for each $k = 1, \ldots, \ell$ either $h_k = 0$, or
      i. $\mathrm{lm}(h_k)\,\mathrm{lm}(g_k) \leq \mathrm{lm}(f)$, and
      ii. $\mathrm{lm}(h_k)\,\mathrm{sig}(g_k) \preceq \mathrm{sig}(f)$.

Remark 2.3.

1. If f reduces sig-safe to g modulo G, then it has a standard representation modulo G. Moreover, note that the concept of sig-safeness, that means the restriction of the reducer g_{j_i} by $t_i \mathrm{sig}(g_{j_i}) \prec \mathrm{sig}(r_{i-1})$ in each step, is essential for the correctness (and the performance) of signature-based algorithms.
2. In Fact 24 of [6] it is shown that it is sufficient to consider signatures with coefficient 1. Thus there is no need to consider $\mathrm{lc}(h_j)$ in Definition 2.1 resp. terms in general for signatures.

The following statement is the signature-based counterpart of Buchberger's Criterion, see [2].

Theorem 2.4. *Let* $G_{i+1} = \{g_1, \ldots, g_\ell\} \subset \mathcal{S} \times I_{i+1}$ *such that* $\{f_1, \ldots, f_{i+1}\} \subset \mathrm{poly}(G_{i+1})$. *If for each pair* (j, k) *with* $j > k$, $1 \leq j, k \leq \ell$, $\mathrm{spoly}(g_j, g_k)$ *has a standard representation w.r.t.* G_{i+1}, *then* $\mathrm{poly}(G_{i+1})$ *is a Gröbner basis of* I_{i+1}.

Proof. For example, see [5,6]. □

In the non-signature-based setting, an algorithm based on the Buchberger Criterion only would be quite inefficient. There the Product Criterion and the Chain Criterion, see [2,12] are used to reduce useless computations; a notable implementation can be found in [10]. On the signature-based side the very same holds: We need criteria to improve the computations, see [6] for more details on this topic.

In the following we assume an incremental signature-based Gröbner basis algorithm, for example, the reader can think of F5's presentation in [7], G2V as stated in [8], or the more general discussion in [6].

Next we state the basic criteria used in F5. It is important to start from F5's point of view to see how optimizations in the signature-based world can be achieved. Note that we use the notations introduced in [6] for an easier adoption to incremental signature-based algorithms in general later on.

Lemma 2.5. *Assume the computation of a Gröbner basis* $\mathrm{poly}(G_i)$ *for* I_i, *that means we are in the* i*th incremental step of* F5.

1. Non-minimal signature criterion (NM)*:* $\mathrm{spoly}(f, g)$ *has a standard representation w.r.t.* G_i *if there exists an element* $\omega \in PSyz(F_i)$ *with* $\mathrm{lm}(\omega) = t_\omega e_{j_h}$ *such that*
   (a) $t_\omega \mid u_h t_h$,

   where $u_h \mathrm{sig}(h) = u_h t_h e_{j_h}$ *for either* $h = f$ *or* $h = g$.
2. Rewritable signature criterion (RW)*:* $\mathrm{spoly}(f, g)$ *has a standard representation w.r.t.* G_i *if there exists a labeled polynomial* r *such that*

(a) $\mathrm{index}(r) = \mathrm{index}(h)$,

(b) $\mathrm{sig}(r) \succ \mathrm{sig}(h)$, *and*

(c) $t_r \mid u_h t_h$,

where $\mathrm{sig}(r) = t_r e_{j_h}$ *and* $u_h \mathrm{sig}(h) = u_h t_h e_{j_h}$ *for either* $h = f$ *or* $h = g$.

Proof. The two statements of Lemma 2.5 are included in Theorem 18 of [5]. Note that there (NM) is considered only for h being an element generated in the current iteration step, that means $\mathrm{index}(h) = i$. Reviewing the proof one easily sees that the situation of $\mathrm{index}(h) < i$ is just a special case already considered by the proof: There the two signatures $u_f \mathrm{sig}(f)$ and $u_g \mathrm{sig}(g)$ refer to $\mathcal{M}$ and $\mathcal{N}$, where $\mathcal{M} \succ \mathcal{N}$. In our situation $u_h \mathrm{sig}(h) = \mathcal{N}$, and any principal syzygy ω with the above mentioned properties can only decrease $\mathcal{N}$. The statement then follows by the very same argumentation as in the proof given in [5]. □

Remark 2.6.

1. Note that all signature-based algorithms have in common that they handle their s-polynomials by increasing signature. This even holds for F5, also there the critical pairs are presorted by the degree of the pairs. Since F5, as presented in [5, 7], works only with homogeneous input, this does not interfere an ordering w.r.t. increasing signatures. By the discussion in [6] F5 can also be used for inhomogeneous input by removing the presorting of critical pairs by increasing degree.

2. F5C and G2V implement (NM) and (RW) quite differently. An in-depth discussion on this topic is given in [6]. These differences are explained in the following, too, inasmuch as they influence the ideas of this paper.

3. The crucial fact for the optimization presented in this paper is that whereas checking (NM) is quite easy and cheap speaking in a computational manner, searching for possible elements r with which we can check (RW) costs many more CPU cycles.

3 Computational overhead

One of the main problems of signature-based Gröbner basis algorithms is the overhead generated by the following kind of data:

1. From the point of view of the resulting Gröbner basis the elements are useless, that means the corresponding leading monomials are superfluous.

2. For the correctness of the algorithm the very same elements are crucial: They are essential for the correct detection of useless critical pairs w.r.t. (NM) and (RW).

Remark 3.1. This characteristic is unique to signature-based algorithms and cannot be found in other Buchberger-style Gröbner basis algorithms. It does not only give a penalty on the performance, but unfortunately also causes problems with theoretical aspects,for example regarding the termination of F5, see [4] for more details.

Next we state the pseudo code of the main loop of an incremental signature-based Gröbner basis algorithm in the vein of F5, denoted SigGB. Note that we denote the incrementally called subalgorithm IncSig.

Let us start the discussion on computational overhead looking at F5 as presented in [7] first, so the following disussions refers to Algorithm 1. The first drawback is the computation of *non-minimal* intermediate Gröbner bases.

1. Due to the fact that the signatures of the labeled polynomials must be kept valid during the (sig-safe) reductions taking place, some leading term reductions do not take place immediately, but are postponed. These reductions, needed to ensure correctness of the algorithm, are computed when generating new critical pairs later on[2]. Thus at the end we could have three polynomials $\mathrm{poly}(f)$, $\mathrm{poly}(g)$, and $\mathrm{poly}(h)$ in $\mathrm{poly}(G_i)$ such that

   (a) $\mathrm{lm}(g) \mid \mathrm{lm}(f)$, but the reduction $f - ctg$ has not taken place due to $t\mathrm{sig}(g) \succ \mathrm{sig}(f)$, for some $c \in \mathcal{K}$, $t \in \mathcal{M}$ such that $\mathrm{lt}(f) = ct\,\mathrm{lt}(g)$.

[2] In [7] this kind of generation of new critical pairs is not postponed to the end of the current reduction step, but those are added to the pair set in place. These two ways of handling such a situation are nearly equivalent and do not trigger any difference for the overall computations, see [6]. The way it is described here makes it easier to see how the computational overhead is produced.

Algorithm 1 SigGB, an incremental signature-based Gröbner basis algorithm in the vein of F5

Input: F_m
Ensure: G, a Gröbner basis for I_m
$G_1 \leftarrow \{(e_1, f_1)\}$
for $(i = 2, \ldots, m)$ **do**
  $f_i \leftarrow$ Reduce $(f_i, \text{poly}(G_{i-1}))$
  if $(f_i \neq 0)$ **then**
    $G_i \leftarrow$ IncSig (f_i, G_{i-1})
  else
    $G_i \leftarrow G_{i-1}$
return $\text{poly}(G_m)$

(b) h is the result of the later on generated and reduced s-polynomial $\text{spoly}(g, f) = ctg - f$, which is sig-safe due to swapping ctg and f.

In the end, we only need two out of these three elements for a Gröbner basis; in a minimal Gröbner basis we would discard $\text{poly}(f)$. The problem is that for the correctness of the ongoing incremental step of F5 the labeled polynomial f as well as its addition to G_i is essential[3]: Without adding f to G_i the critical pair (g, f) would not be generated at all, thus the element h, possibly needed for the correctness of the Gröbner basis in the end, would never be computed. So we are not able to remove f during the actual iteration step.

Clearly, in the same vein the problem of non-reducedness of the Gröbner basis $\text{poly}(G_i)$, in particular, missing tail-reductions, can be understood.

2. Since F5, when reducing with elements generated in the ongoing iteration step, processes top-reductions only, elements can enter G_i whose polynomials have tails not reduced w.r.t. $\text{poly}(G_i)$. The main argument for not doing complete reductions in this situation is the requirement of sig-safeness: Comparing the signatures before each possible tail-reduction can lead to quite worse timings. On the other hand, from the point of view of the resulting Gröbner basis $\text{poly}(G_i)$, which consists only of polynomial data, we do not need to take care of sig-safeness and can tail-reduce the elements in $\text{poly}(G_i)$ as usual without any preprocessed signature comparison. This is way faster than implementing tail-reductions during the iteration step, although we have to use the non-tail-reduced elements during a whole iteration step.

From the above discussion we get the following situation:

1. The computational overhead *during* an iteration step is prerequisite for the correctness of F5.
2. The set of labeled polynomials G_i returned *after* the ith iteration step is used as input for the $(i+1)$st iteration step, including the signatures.

In [13] Stegers found a way optimizing at least the reduction steps w.r.t. elements of previous iteration steps. There the fact is used that F5 does not need to look for the signatures, due to the definition of $\prec$ all such reducers have a smaller index, and thus, a smaller signature: His variant of F5 computes another set of polynomials B_i after each iteration step, namely the reduced Gröbner basis of I_i which is computed out of $\text{poly}(G_i)$. In the following iteration step reductions w.r.t. elements computed in previous iteration steps are done by B_i, not by $\text{poly}(G_i)$.

In [5] the variant F5C of F5 is presented, which is based on the idea of Stegers, but goes way further: F5C is an optimized version of F5 interreducing the intermediate Gröbner basis $\text{poly}(G_i)$ to B_i and uses these *polynomial data* as starting point for the next iteration step. At this point we can look at Algorithm 1 from [6], which illustrates one single iteration step of incremental signature-based Gröbner basis algorithms: Let $\ell = \#B_i$, any element $b_j \in B_i$ gets a new signature e_j, so that we receive elements $g_j = (e_j, b_j)$ in G_{i+1} for $1 \leq j \leq \ell$. f_{i+1} is then added to G_{i+1} by adjusting the index, $g_{\ell+1} = (e_{\ell+1}, f_{i+1})$. On the one hand, proceeding this way the corresponding signatures of reduced polynomials are guaranteed to be correct from the view of the algorithm. On the other hand, all previously available criteria for detecting useless critical pairs w.r.t. to labeled polynomials of index $\leq \ell$ by (RW) in the

[3] See [4] for more details, also on termination issues caused by this behaviour of signature-based algorithms.

upcoming iteration step are removed. Thus the question, if for the benefit of having less computational overhead is payed dearly by less efficient criteria checks in the following iteration steps, needs to be asked.

The idea to get at least some data for using (RW) on elements of previous iteration steps in the following can be summarized in this way:

1. For any two elements $g_j, g_k \in G_{i+1}$ with polynomial parts in B_i, $\operatorname{spoly}(g_j, g_k)$ reduces to zero w.r.t. G_{i+1}, that means, it already has a standard representation w.r.t. G_{i+1} (since B_i is already a reduced Gröbner basis). Moreover, we can quite easily get the corresponding signatures of $\operatorname{spoly}(g_j, g_k)$ due to the fact that the labels of the generating elements, e_j and e_k, are trivial.

2. With this in mind, we can create for each index $2 \leq j \leq \ell$ a list of signatures generated out of $\operatorname{sig}(\operatorname{spoly}(g_j, g_k))$ for $k < j$. Those signatures can be used by (RW) detecting useless critical pairs in the next iteration step.

Clearly, one wants to omit such a recomputation of signatures during two iteration steps as much as possible. Luckily it is shown in [5] that this is not a problem at all:

Proposition 3.2. *Let* $\operatorname{spoly}(f, g)$ *be any s-polynomial considered during an iteration step of* F5 *with* $\operatorname{index}(g) < \operatorname{index}(f)$. *Assume that* $\frac{\tau(f,g)}{\operatorname{lm}(g)} g$ *would be detected either by (NM) or (RW). Then* $\operatorname{spoly}(f, g)$ *is also discarded in* F5C.

Corollary 3.3. *For an s-polynomial in* F5C *it is enough to check a generator* f *by (NM) resp. (RW) if* f *was computed during the current iteration step.*

Proof. See Theorem 27 resp. Corollary 28 in [5]. □

Thus it follows that we do not need to recompute any signature after interreducing the intermediate Gröbner basis $\operatorname{poly}(G_i)$ for checks with (RW).

Let us add the above ideas in the pseudo code of Algorithm 2. We highlight the new step of interreducing the intermediate Gröbner basis, differing from the description of Algorithm 1. There are two main changes:

1. G_i now denotes already a set of polynomials in $\mathcal{R}$, as the labeled polynomials are newly generated at the beginning of each incremental step.

2. Instead of IncSig we use a new algorithm IncSigR which takes a reduced Gröbner basis B_{i-1} as a second argument. Note that for IncSigR we refer the reader to Algorithm 1 in [6].

Algorithm 2 SigGB with reduced intermediate Gröbner bases

Input: F_m
Ensure: G, a Gröbner basis for I_m

```
1: G_1 ← {f_1}
2: for (i = 2, ..., m) do
3:    B_{i-1} ← RedSB (G_{i-1})
4:    f_i ← Reduce(f_i, B_{i-1})
5:    if (f_i ≠ 0) then
6:       G_i ← IncSigR(f_i, B_{i-1})
7:    else
8:       G_i ← G_{i-1}
9: return G_m
```

Remark 3.4.

1. Also we do not state proofs for Proposition 3.2 and Corollary 3.3 here, but only refer to [5], it is important to note that correctness of Proposition 3.2 is based on the fact that F5C uses (RW) as presented above. G2V uses a relaxed variant of (RW), see [6] for more details. Still, G2V uses Algorithm 2 as an outer loop for its incremental computations.

2. In the following we can consider incremental signature-based Gröbner basis algorithms in general, i.e., Algorithm 2 can be seen as a wrapper for F5C and G2V.

Having a common basis of our algorithms in question, let us see next how the initial presentation of G2V improved the field of signature-based computations.

4 Using reductions to zero

As described in [6] G2V can be seen a variant of F5C using a way more relaxed version of (RW): G2V only checks if the corresponding s-polynomials of two critical pairs have the same signature when adding the pairs to the pair set. In this situation only one of these two pairs is kept, the other one is discarded. We refer to [6] for more details.

Thus G2V's efficiency is mainly based on its optimized variant of (NM). The idea can be explained quite easily: Whereas F5C uses only principal syzygies for (NM), since they are known beforehand and can be precomputed, G2V goes one step further:

Definition 4.1. During the $(i+1)$st iteration step of IncSigR we define

$$S_{i+1} := \{te_{\ell+1} \in \mathcal{S} \mid te_{\ell+1} \text{ signature of an s-polynomial} \\ \text{that reduced sig-safe to zero }\},$$

where $\ell+1$ is the current index of the labeled polynomials.

Lemma 4.2 (Improved (NM))**.** *Let $\ell+1$ be the current index of the $(i+1)$st incremental step of* G2V *computing a Gröbner basis* $\operatorname{poly}(G_{i+1})$ *for I_{i+1}.* $\operatorname{spoly}(f,g)$ *has a standard representation w.r.t. G_{i+1} if there exists an element $\omega \in PSyz(B_i \cup \{f_{i+1}\}) \cup S_{\ell+1}$ with* $\operatorname{lm}(\omega) = t_\omega e_{\ell+1}$ *such that $t_\omega \mid u_h t_h$, where* $u_h \operatorname{sig}(h) = u_h t_h e_{\ell+1}$ *for either $h = f$,* $\operatorname{index}(f) = \ell+1$ *or $h = g$,* $\operatorname{index}(g) = \ell+1$.

Proof. See Proposition 16 and Lemma 17 in [6]. □

Remark 4.3.

1. Switching from $\operatorname{PSyz}(F_{i+1})$ to $\operatorname{PSyz}(B_i \cup \{f_{i+1}\})$ is not a problem at all since $\langle F_{i+1}\rangle = \langle B_i \cup \{f_{i+1}\}\rangle$ as B_i is the reduced Gröbner basis of I_i.
2. Note that the restriction to check elements of current index only is influenced by the discussion in Section 3. Whereas we know that Corollary 3.3 ensures that F5C does not lose any useful information for rejecting useless critical pairs due to its aggressive implementation of (RW), this does not hold for G2V. So for G2V it is possible that removing the signatures of the intermediate Gröbner bases can lead to the situation that less critical pairs are rendered useless by (NM).

Clearly, one can easily use (NM) as given in Lemma 4.2 in F5C instead of the plain one stated in Lemma 2.5, its correctness does not depend on the implementation of (RW).

Definition 4.4. We denote the algorithm F5C with (NM) implemented as in Lemma 4.2 by F5A.[4]

The question that comes to one's mind is the following: Why would it be of any benefit to switch from F5C to F5A? Whereas some of the signatures of zero reductions are used in F5C in (RW), not all of them can be used in each situation due to the restriction that $\operatorname{sig}(r) \succ \operatorname{sig}(h)$. It is already mentioned in [6] that F5A is way faster than F5C for non-regular input: F5C computes way more zero reductions than F5A, but cannot use it actively. Moreover, even if a corresponding (RW) detection happens in F5C, testing by (NM) in F5A is a lot faster as already discussed in Remark 3. We see in Section 6 that this has a huge impact not only on the timings, but also on the number of reduction steps taking place, due to the fact that (NM) and (RW) are used in F5C resp. F5A on possible reducers, too.

5 Combining ideas

So what is our contribution in this paper besides giving an overview of already known optimizations? Until now, the presented ideas of interreducing intermediate Gröbner bases and using zero reductions actively in (NM) are used without any direct connection:

1. The Gröbner bases are interreduced *between two iteration steps.* This has an effect on the labeled polynomials computed in the previous iteration steps.

[4] The "A" stands for "actively using zero reductions".

2. Zero reductions are used actively *in a single iteration step only.* This has an impact on current index labeled polynomials only.

Of course, interreducing the intermediate Gröbner basis $\operatorname{poly}(G_i)$ to $B_i$has an influence on the upcoming iteration step inasmuch as less critical pairs are considered and reductions w.r.t. B_i are more efficient. Besides this we cannot assume to receive any deeper impact on the $(i+1)$st iteration step. On the other hand, it would be quite nice to use (NM) not only on current index labeled polynomials, but also on those coming from B_i. For this one could just precompute $\operatorname{PSyz}(B_i)$ and check the corresponding lower index generators of critical pairs using $\operatorname{PSyz}(B_i)$ in (NM). By Lemma 2.5 this would be a correct optimization. The crucial point is that we can do even better:

At the moment we interreduce the intermediate Gröbner basis $\operatorname{poly}(G_i)$ of I_i let us try to get some more resp. better signatures for checking (NM): We have already seen in Section 3 how to recompute some signatures for checking lower index labeled polynomials in the upcoming iteration step: We just compute and keep the signatures of the corresponding s-polynomials $\operatorname{spoly}(g_j, g_k)$ for $\operatorname{poly}(g_j), \operatorname{poly}(g_k) \in B_i$. There we could reject their computations as we have proven that F5C would discard any s-polynomial detected with those signatures by (RW) anyway. Moreover, we know that all those signatures correspond to s-polynomials that already reduce to zero w.r.t. B_i. Thus, with the idea of Section 4 in mind, we can actively use those theoretical zero reductions actively for lower index labeled polynomials.

Definition 5.1. Assume that $\operatorname{poly}(G_i)$ is reduced to $B_i = \{b_1, \ldots, b_{\ell_i}\}$ after the ith iteration step of SigGB. Then we define

$$S_i := \left\{ \frac{\tau(b_{\ell_i}, b_k)}{\operatorname{lm}(b_{\ell_i})} e_{\ell_i} \mid 1 \leq k < \ell_i \right\}.$$

Theorem 5.2 (Strengthening (NM))**.** *Assuming the* $(i+1)$*st incremental step of a signature-based algorithm computing a Gröbner basis* $\operatorname{poly}(G_{i+1})$ *for* I_{i+1}*, let* $\ell+1$ *be the current index of labeled polynomials.* $\operatorname{spoly}(f,g)$ *has a standard representation w.r.t.* G_{i+1} *if there exists an element*

$$\omega \in \mathit{PSyz}(B_i \cup \{f_{i+1}\}) \cup S_{i+1} \cup S_i \cup \ldots \cup S_2$$

with $\operatorname{lm}(\omega) = t_\omega e_{j_h}$ *such that* $t_\omega \mid u_h t_h$*, where* $u_h \operatorname{sig}(h) = u_h t_h e_{j_h}$ *for either* $h = f$ *or* $h = g$*.*

Proof. Clearly, the elements ω in $\operatorname{PSyz}(B_i \cup \{f_{i+1}\})$ can be divided in two types:

1. $\operatorname{lm}(\omega) = te_{\ell+1}$, and
2. $\operatorname{lm}(\omega) = te_j$ for $2 \leq j < \ell+1$.

Those elements of Type (1) from $\operatorname{PSyz}(B_i \cup \{f_{i+1}\})$ together with those of S_{i+1} (see Definition 4.1) are of the type $te_{\ell+1}$, and thus do only detect generators of the s-polynomial in question which are computed during the current iteration step. The correctness of this statement follows from Lemma 4.2.

Next we consider S_j for $2 \leq j \leq i$: Note that each element in such an S_j is of type te_{ℓ_j}, that means those elements can only detect generators of the s-polynomial in question of index ℓ_j. Thus it is enough to consider one such S_j for a fixed j. Let $te_{\ell_j} \in S_j$, then $t = \frac{\tau(b_{\ell_j}, b_k)}{\operatorname{lm}(b_{\ell_j})}$ for some $1 \leq k < \ell_j$. Since B_j with $\#B_j = \ell_j$ is a reduced Gröbner basis we know that $\operatorname{spoly}(b_{\ell_j}, b_k)$ reduces to zero w.r.t. B_j. Moreover, note that due to the incremental structure of SigGB and the interreduction of $\operatorname{poly}(G_j)$ between each two iteration steps the sequence $(b_1, \ldots, b_{\ell_j})$ of elements of B_j can be ordered in such a way that this zero reduction corresponds to a syzygy ω with $\operatorname{lm}(\omega) = te_{\ell_j}$. Any other possible syzygy detecting a generator of index ℓ_j is of Type (2) from $\operatorname{PSyz}(B_i \cup \{f_{i+1}\})$. By Lemma 2.5 (1) the statement holds. □

Remark 5.3.

1. Note that the sets S_i in Theorem 5.2 are computed recursively after the ith iteration step of SigGB. There is no connection between S_i and S_{i-1} for any $i > 3$. It is also crucial to only take the element of highest index ℓ_i from B_i into account when computing S_i, we cannot ensure that $\operatorname{spoly}(g_j, g_k)$ reduces sig-safe to zero w.r.t. G_i if $j \neq \ell_i \neq k$. There a not sig-safe reduction could be possible to achieve the zero reduction of $\operatorname{spoly}(g_j, g_k)$, a problem that cannot happen if either j or k is equal to ℓ_i.

2. The main optimization compared to F5C is to use S_i up to S_2 not in (RW), as it is described in Section 3, but in (NM). Compared to G2V's variant of (NM) lots of new checks are added that detect way more useless critical pairs as we see in Section 6.

   Note that this problem has not been taken into account in [5] where such signatures are suggested to be used in (RW).

One could think that the presented strengthening of (NM) is rather equivalent to the initial presentation in Lemma 2.5, but this is not the fact. The variant presented here is way more aggressive in finding useless critical pairs.

Corollary 5.4. *In Theorem 5.2 elements of type* $\operatorname{lm}(s) = te_j \in \mathit{PSyz}(B_i \cup \{f_{i+1}\})$ *where* $j = \#B_\ell$ *for some* $2 \le \ell \le i$ *is need not be considered at all.*

Proof. Assume such an $s \in \operatorname{PSyz}(B_i \cup \{f_{i+1}\})$ with $\operatorname{lm}(s) = te_j$, $j = \#B_\ell$ for some fixed ℓ. Then $s = b_k e_j - b_j e_k$ for some $k < j$. Let k be fixed. In S_ℓ there exists some ue_j with $u = \frac{\tau(b_j,b_k)}{\operatorname{lm}(b_j)}$. It follows that $u \mid t$. □

The idea is now to replace the implementations of (NM) in F5C, F5A, and G2V , respectively, by the version given in Theorem 5.2.

Definition 5.5. We denote the algorithms F5C, F5A, and G2V with (NM) implemented as in Theorem 5.2 by iF5C, iF5A, and iG2V, respectively.[5]

Algorithm 3 illustrates the main wrapper for an incremental signature-based Gröbner basis algorithm based on the idea presented in this section.

Algorithm 3 SigGB with reduced intermediate Gröbner bases and optimized (NM) Criterion

Input: F_m
Ensure: G, a Gröbner basis for I_m

$S \leftarrow []$, $G_1 \leftarrow \{f_1\}$
for $(i = 2, \ldots, m)$ **do**
  $B_{i-1} \leftarrow \textsc{RedSB}\,(G_{i-1})$
  $f_i \leftarrow \textsc{Reduce}(f_i, B_{i-1})$
  if $(f_i \neq 0)$ **then**
    $j \leftarrow \#B_{i-1}$
    for $(k = 1, \ldots, j-1)$ **do**
      $S_{i-1,k} \leftarrow \frac{\tau(b_j,b_k)}{\operatorname{lm}(b_j)} e_j$
    $G_i \leftarrow \textsc{IncSigR}(f_i, B_{i-1})$
  else
    $G_i \leftarrow G_{i-1}$
return G_m

Example 5.6. Let $p_1 = yz + 2$, $p_2 = xy + \frac{1}{3}xz + \frac{2}{3}$, $p_3 = xz^2 - 6x + 2z$ be three polynomials in $\mathbb{Q}[x, y, z]$. We want to compute a Gröbner basis for the ideal $I = \langle p_1, p_2, p_3\rangle$ w.r.t. the degree reverse lexicographical ordering with $x > y > z$ using Algorithm 3 using $\prec$ as ordering on the signatures.

Due to the incremental structure of the algorithm we start with the computation of a Gröbner basis G_2 of $\langle p_1, p_2\rangle$. After initializing $f_1 := (e_1, p_1)$ and $f_2 := (e_2, p_2)$ we construct the s-polynomial of f_1 and f_2 is

$$\operatorname{spoly}(f_2, f_1) = (ze_2, zp_2 - xp_1),$$

which does not have a standard representation w.r.t. G_2 at the moment of its creation. Nevertheless, a new element is added to G_2, namely

$$f_3 := \left(ze_2, \frac{1}{3}xz^2 - 2x + \frac{2}{3}z\right).$$

[5] The "i" stands for "intermediate incremental optimization".

Now we look at the s-polynomials

$$\begin{aligned}\operatorname{spoly}(f_3, f_1) &= (yze_2, 3y \operatorname{poly}(f_3) - xz \operatorname{poly}(f_1)),\\ \operatorname{spoly}(f_3, f_2) &= \left(yze_2, 3y \operatorname{poly}(f_3) - z^2 \operatorname{poly}(f_2)\right).\end{aligned}$$

It does not make any difference which of these two s-polynomials we compute: F5 would remove the later one by its implementation of (RW), whereas G2V would only store one of the two corresponding critical pairs in the beginning. W.l.o.g. we assume the reduction of $\operatorname{spoly}(f_3, f_1)$, the situation for considering $\operatorname{spoly}(f_3, f_2)$ is similar and behaves the very same way:

$$\operatorname{spoly}(f_3, f_1) = (yze_2, -6xy - 2xz + 2yz)$$

Further sig-safe reductions with $6f_2$ and $2f_1$ lead to a zero reduction, i.e. we can add a new rule, namely yze_2, to the set S_2 as explained in Section 4.[6]

Since there is no further s-polynomial left, SigGB finishes this iteration step with

$$G_2 = \left\{ yz + 2, xy + \frac{1}{3}xz + \frac{2}{3}, \frac{1}{3}xz^2 - 2x + \frac{2}{3}z \right\}.$$

We see that G_2 is already the reduced Gröbner basis B_2 of $\langle p_1, p_2 \rangle$, so Line 3 of SigGB does not change anything. Since G_i and B_i do not coincide inbetween most iteration steps, we need to remove S_2 completely, since rules stored in there might not be correct any more.

Let us have a closer look at how the new rules would be computed: First of all, any element in G_2 corresponds to a module generator of $\mathcal{R}^3$, that means we can think of three labeled polynomials $f_i := (e_i, b_i)$ for $i \in \{1, 2, 3\}$. During the lines 6 – 8 the signatures corresponding to $\operatorname{spoly}(f_3, f_1)$ and $\operatorname{spoly}(f_3, f_2)$ are added to S_2 respectively: The first one gives the rule ye_3, the second one also; thus we have $S_2 = \{ye_3\}$.

Next we see that p_3 reduces to zero w.r.t. B_2 in Line 4.

Thus the algorithm does not enter another iteration step, but terminates with the correct Gröbner basis $G = B_2$.

Remark 5.7.

1. Of course as stated in the pseudo code, SigGB would reduce the next generator p_3 of the input ideal w.r.t. intermediate reduced Gröbner basis B_2, *before* it would recompute the syzygy list S_2 in the above example. For the sake of explaining how the recomputation of such a rules list works, using a rather small example, we choose a complete discussion over efficiency in Example 5.6

2. Note that it would be wrong to also add the signature of $\operatorname{spoly}(f_2, f_1)$ to S_2 once we have interreduced G_2 to B_2. Of course, $\operatorname{spoly}(b_2, b_1)$ has a standard representation w.r.t. B_2, namely

$$\operatorname{spoly}(b_2, b_1) = b_3,$$

   but this does not lead to a standard representation of $\operatorname{spoly}(f_2, f_1)$ due to the fact that $\operatorname{sig}(f_3) = e_3 \succ \operatorname{sig}(\operatorname{spoly}(f_2, f_1))$. That is the one big drawback of interreducing the intermediate Gröbner bases in incremental signature-based algorithms. Nevertheless, the speed up due to handling way less elements in B_i compared to G_i more than compensates this as shown in [5].

6 Experimental results

We compare timings, the number of zero reductions, and the number of overall reduction steps of the different algorithms presented in this paper. To give a faithful comparison, we use a further developed version of the implementation we have done for [6]: This is an implementation of a generic signature-based Gröbner basis algorithm in the kernel of a developer version of Singular 3-1-4. Based on this version we implemented G2V, iG2V, F5C, iF5C, F5A, and iF5A by plugging in the different variants and usages of the criteria (NM) and (RW). There are no optimizations which could prefer any of the specific algorithms, so that the difference in the implementation between two of the above mentioned algorithms is not more than 300 lines of code; compared to approximately 3,500 lines

[6]Latest at this point $\operatorname{spoly}(f_3, f_2)$ would be removed by the rule $yze_2 \in S_2$.

Figure 6.1: Meanings of the colors in the respective tables

of code overall this is negligible. All share the very same data structures and use the very same (sig-safe) reduction routines. So the differences shown in Tables 1, 2, and 3 really come from the various optimizations of the criteria mentioned in Sections 3 – 5.

Of course, to ensure such an accurate comparison of various different variants of signature-based algorithms has a drawback in the overall performance of the algorithms. Since we are interested in impact of the improvements explained in this paper it is justified to take such an approach. Clearly, implementing a highly optimized iF5A whithout any restrictions due to sharing data structures and procedures with an G2V can lead to a way better performance. It is not in focus of this paper to present the fastest implementation of such kind of algorithms, but to present practical benefits of the presented optimizations, focusing on the fact that all variants of incremental signature-based Gröbner basis algorithms take an advantage out of them.

The source code is publicly available in the branch `f5cc`[7] at

`https://github.com/ederc/Sources`.

We computed the examples on a computer with the following specifications:

- 2.6.31–gentoo–r6 GNU/Linux 64–bit operating system,
- INTEL® XEON® X5460 @ 3.16GHz processor,
- 64 GB of RAM, and
- 120 GB of swap space.

Due to the fact that we are comparing 6 algorithms we colorized the results presented in the respective tables for better readability, see Figure 6.1.

Our naming convention for examples specifies that "`-h`" denotes the homogenized variant of the corresponding benchmark. All examples are computed over a field of characteristic $32,003$ w.r.t. the degree reverse lexicographical ordering.

First of all let us have a closer look at G2V, F5C, and F5A. In Table 1, we see that whereas F5C is faster than G2V in nearly all example sets, the ones which lead to a high number of zero reductions in F5C are computed way faster by G2V. This is based on the fact that G2V actively uses such zero reductions with adding new checks for (NM), whereas F5C only partially includes those signatures in its implementation of (RW).

Comparing F5A and F5C we see that F5A is not only way faster than F5C in such highly non-regular examples like `Eco-X`[8], but also that F5A is faster in other systems like `Cyclic-8`. Note that F5A is in all examples faster and computes less reductions than G2V.

The ideas of Section 5 help iG2V to compute much less reduction steps and discard way more useless critical pairs than G2V does. This comes from the fact that G2V does not implement any rewritable criterion besides its choice of keeping only 1 critical pair per signature. In most examples iG2V executes only half as much reduction steps as G2V, in some examples like `F-855` it even alleviates to 15%.

By our discussion in Section 3 it is not a surprise that there is no change in the corresponding numbers of reduction steps and zero reductions for iF5C resp. iF5A compared to the ones of F5C and F5A. Still the timings improve greatly which is based on the following facts:

1. Altough the rules added to S_i in Lines 6 – 8 are also checked by F5's (RW) implementation, checking (RW) costs more time than checking (NM) due to checking if the one is still in the area of correct (RW) rules.

[7] The results presented here are done with the commit key `5c4dc1134a4ab630faab994dbe93d3013b4ccc7e`.
[8] Note that F5C and iF5C cannot compute `Eco-11-h`.

Test case	G2V	iG2V	F5C	iF5C	F5A	iF5A
`Cyclic-7-h`	25.714	13.441	5.369	5.255	5.790	5.180
`Cyclic-7`	24.613	12.782	5.660	5.310	5.216	4.758
`Cyclic-8-h`	13,273.487	5,710.221	6,881.895	3,689.003	4,970.714	1,907.076
`Cyclic-8`	12,386.925	5,362.307	6,606.116	3,482.177	4,814.173	1,812.321
`Eco-9-h`	10.050	6.741	58.935	59.182	6.320	5.550
`Eco-9`	18.755	10.012	12.899	12.994	13.425	12.899
`Eco-10-h`	278.663	158.167	2,472.763	1,974.163	188.683	133.183
`Eco-10`	1,041.896	497.025	491.914	518.346	492.345	497.811
`Eco-11-h`	10,169.897	5,136.465	–	–	7,893.788	4,815.971
`F-744-h`	33.244	25.724	35.324	36.740	19.865	19.865
`F-744`	27.312	20.469	8.198	9.267	8.458	8.198
`F-855-h`	1,246.744	290.856	2,948.138	2,381.813	600.001	422.219
`F-855`	971.491	131.134	83.185	86.343	86.558	84.020
`Katsura-10-h`	4.186	4.193	4.213	4.491	4.248	4.256
`Katsura-10`	4.150	4.183	4.187	4.217	4.227	4.192
`Katsura-11-h`	59.004	59.871	58.689	61.496	58.411	58.673
`Katsura-11`	53.894	53.855	53.464	56.122	53.984	53.118
`Gonnet-83-h`	12.165	10.617	126.173	25.963	9.811	8.761
`Schrans-Troost-h`	4.393	4.250	2.970	3.498	3.087	2.970

Table 1: Time needed to compute a Gröbner basis of the respective test case, given in seconds.

2. To have all possible (RW) rules available, (RW) must be checked directly before the reduction step of the corresponding critical pair starts. At this point the critical pair can be stored for a long time, using memory and making the list of critical pairs longer. In iF5C resp. iF5A useless critical pairs can be found directly when the algorithm tries to create them. Thus they are not kept for a long time, keeping lists shorter, which does not only save memory, but also speeds up insertion of other critical pairs to the list.

All three variants implementing the idea of Section 5 are, often multple times, faster in all examples, besides `Katsura-X`; for them we already know that the usual implementation of (NM), that means considering the principal syzygies, is already optimal. Thus all the ideas presented in this paper only add some small overhead in the computations of `Katsura-X` which does not affect the performance in a beneficial way.

7 Conclusion

This paper contributes a more efficient usage, generalization, and combination of, for specific incremental signature-based algorithms, known improvements. Even in situations where it does not enlarge the number of detected useless critical pairs (F5C, F5A) it gives quite impressive speed-ups using a faster, less complex way of detecting useless data.

Looking explicitly at G2V, the improvement in terms of removing redundant critical pairs is astonishing. Due to the fact that G2V lacks a real implementation of (RW) the idea presented in Section 5 gives an easy way to add, at least partly, the strengths of F5's (RW) implementation to G2V without making the algorithm's description more complex.

We have presented a rather neat improvement that can be added to any existing implementation of an incremental signature-based algorithm without any bigger effort. Besides having a huge impact on the computations for non-regular sequences, it does not slow down the overall efficiency of incremental signature-based Gröbner basis algorithms for regular input at all.

Test case	G2V	iG2V	F5C,iF5C	F5A,iF5A
Cyclic-7-h	1,750,989	625,815	100,569	83,880
Cyclic-7	1,750,989	625,815	100,569	83,880
Cyclic-8-h	113,833,183	44,663,466	14,823,873	3,403,874
Cyclic-8	113,833,183	44,663,466	14,823,873	3,403,874
Eco-9-h	409,880	238,841	1,996,849	136,842
Eco-9	551,837	310,745	247,434	247,434
Eco-10-h	3,760,244	1,996,573	19,755,560	1,019,439
Eco-10	6,853,713	3,352,474	2,384,889	2,384,889
Eco-11-h	33,562,613	16,695,766	-	7,374,779
F-744-h	1,082,448	693,630	789,072	435,869
F-744	473,838	285,402	179,100	179,100
F-855-h	23,097,574	4,407,938	12,294,951	2,633,666
F-855	7,976,163	1,772,726	835,718	835,718
Katsura-10-h	18,955	18,955	18,343	18,343
Katsura-10	18,955	18,955	18,343	18,343
Katsura-11-h	65,991	65,991	63,194	63,194
Katsura-11	65,991	65,991	63,194	63,194
Gonnet-83-h	113,609	93,137	278,419	93,137
Schrans-Troost-h	19,132	18,352	14,010	14,010

Table 2: Number of all reduction steps throughout the computations of the algorithms.

Test case	G2V,iG2V	F5C,iF5C	F5A,iF5A
Cyclic-7-h	36	76	36
Cyclic-7	36	76	36
Cyclic-8-h	244	1,540	244
Cyclic-8	244	1,540	244
Eco-9-h	120	929	120
Eco-9	0	0	0
Eco-10-h	247	2,544	247
Eco-10	0	0	0
Eco-11-h	502	-	502
F-744-h	323	498	323
F-744	0	0	0
F-855-h	835	2,829	835
F-855	0	0	0
Katsura-10-h	0	0	0
Katsura-10	0	0	0
Katsura-11-h	0	0	0
Katsura-11	0	0	0
Gonnet-83-h	2,005	8,129	2,005
Schrans-Troost-h	0	0	0

Table 3: Number of zero reductions computed by the algorithms.

Acknowledgements

I would like to thank the referees for improving this paper with their suggestions, especially the one pointing out an error in a previous version of this paper, a fact Example 5.6 mirrors. Furthermore, I would like to thank John Perry for helpful discussions and comments.

References

[1] Arri, A. and Perry, J. The F5 Criterion revised. 2011. http://arxiv.org/abs/1012.3664v3.

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

[3] Decker, W., Greuel, G.-M., Pfister, G., and Schönemann, H. SINGULAR *3-1-4 — A computer algebra system for polynomial computations*, 2012. http://www.singular.uni-kl.de.

[4] Eder, C., Gash, J., and Perry, J. Modifying Faugère's F5 Algorithm to ensure termination. *ACM SIGSAM Communications in Computer Algebra*, 45(2):70–89, 2011. http://arxiv.org/abs/1006.0318.

[5] Eder, C. and Perry, J. F5C: A Variant of Faugère's F5 Algorithm with reduced Gröbner bases. *Journal of Symbolic Computation, MEGA 2009 special issue*, 45(12):1442–1458, 2010. dx.doi.org/10.1016/j.jsc.2010.06.019.

[6] Eder, C. and Perry, J. Signature-based Algorithms to Compute Gröbner Bases. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation*, pages 99–106, 2011.

[7] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In *ISSAC'02, Villeneuve d'Ascq, France*, pages 75–82, July 2002. Revised version from http://fgbrs.lip6.fr/jcf/Publications/index.html.

[8] Gao, S., Guan, Y., and Volny IV, F. A New Incremental Algorithm for Computing Groebner Bases. *Journal of Symbolic Computation – ISSAC 2010 Special Issue*, 1:13–19, 2010.

[9] Gao, S., Volny IV, F., and Wang, D. A new algorithm for computing Groebner bases. 2010.

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

[11] Greuel, G.-M. and Pfister, G. *A* SINGULAR *Introduction to Commutative Algebra*. Springer Verlag, 2nd edition, 2007.

[12] Kollreider, C. and Buchberger, B. An improved algorithmic construction of Gröbner-bases for polynomial ideals. *SIGSAM Bull.*, 12:27–36, May 1978.

[13] Stegers, T. Faugère's F5 Algorithm revisited. Master's thesis, Technische Univerität Darmstadt, revised version 2007.

[14] Sun, Y. and Wang, D. A generalized criterion for signature related Gröbner basis algorithms. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation*, pages 337–344, 2011.