J Syst Sci Complex (2019) 32: 205–233

Speeding Up the GVW Algorithm via a Substituting Method*

LI Ting · SUN Yao · HUANG Zhenyu · WANG Dingkang · LIN Dongdai

DOI: 10.1007/s11424-019-8345-3
Received: 12 September 2018 / Revised: 10 December 2018

Abstract The GVW algorithm is an efficient signature-based algorithm for computing Gröbner bases. In this paper, the authors consider the implementation of the GVW algorithm by using linear algebra, and speed up GVW via a substituting method. As it is well known that, most of the computing time of a Gröbner basis is spent on reductions of polynomials. Thus, linear algebraic techniques, such as matrix operations, have been used extensively to speed up the implementations. Particularly, one-direction (also called signature-safe) reduction is used in signature-based algorithms, because polynomials (or rows in matrices) with larger signatures can only be reduced by polynomials (rows) with smaller signatures. The authors propose a new method to construct sparser matrices for signature-based algorithms via a substituting method. Specifically, instead of only storing the original polynomials in GVW, the authors also record many equivalent but sparser polynomials at the same time. In matrix construction, denser polynomials are substituted by sparser equivalent ones. As the matrices get sparser, they can be eliminated more efficiently. Two specifical algorithms, Block-GVW and LM-GVW, are presented, and their combination is the Sub-GVW algorithm. The correctness of the new proposed method is proved, and the experimental results demonstrate the efficiency of this new method.

Keywords Gröbner basis, GVW, signature-based algorithm, time-memory tradeoff.

LI Ting
SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, *China*; *School of Cyber Security, University of Chinese Academy of Sciences, Beijing* 100190, *China*. Email: liting@iie.ac.cn.
SUN Yao (Corresponding author) · HUANG Zhenyu
SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, *China*.
Email: sunyao@iie.ac.cn; huangzhenyu@iie.ac.cn.
WANG Dingkang
KLMM, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100090, *China*; *School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing* 100190, *China*.
Email: dwang@mmrc.iss.ac.cn.
LIN Dongdai
SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, *China*.
Email: ddlin@iie.ac.cn.

*This research is supported by the National Nature Science Foundation of China under Grant Nos. 61877058, 61872359, the Strategy Cooperation Project under Grant No. AQ-1701, and the CAS Project under Grant No. QYZDJ-SSW-SYS022.
◇ *This paper was recommended for publication by Editor LI Hongbo.*

1 Introduction

Gröbner bases have become important tools in many aspects of applications. Since Gröbner basis is proposed in 1965[1], many efforts have been made to improve its computing efficiency. There are two main kinds of improvements. The first one is to speed up the basic operations during computing Gröbner bases. One landmark improvement is that Lazard pointed out the strong relation between Gröbner bases and linear bases in vector spaces[2]. This idea has been implemented in F4 by Faugère[3], and also in XL type algorithms by Courtois, et al.[4] and Ding, et al.[5]. The second kind improvement tries to detect redundant computations as many as possible. One important event is that Faugère introduced the first signature-based algorithm F5[6]. F5 uses signature-based criteria and is available to reject almost all redundant computations if the input system is semi-regular. After F5 is proposed, many variants have been presented, including F5C[7], extended F5[8], F5 with revised criterion (the AP algorithm)[9], and RB[10]. Gao, et al.[11] proposed another signature-based algorithm G2V in a different way from F5, and GVW[12] which is an extended version of G2V. The generalized criteria and signature-based algorithms in solvable polynomial algebra is studied in [13, 14] as well. For an overview of all signature-based algorithms, please refer to[15].

In the field of implementing signature-based algorithms, Faugère presented his implementation of F5 in[6], and improved it via parallel techniques in [16]. Albrecht and Perry provided an F5 algorithm in F4 style[17]. Matrix-F5 was described in [18, 19]. Roune and Stillman give an efficient implementation of GVW and AP without using linear algebra[20]. Boyer, et al.[21] implemented and improved the idea of Faugère and gave a new GPLv2 open source C library GBLA. The authors implemented GVW in F4 style over Boolean polynomial rings[22] using routines modified from M4RI[23] and also presented a fast method of implementing the Symbolic Preprocessing function[24].

In this paper, we present a new method for speeding up the implementation of the GVW algorithm via a substituting technique. Besides storing polynomials with signatures like in general signature-based algorithms, we also maintain a set of equivalent and sparser polynomials for each general polynomial. For instance, let $(x^\alpha \boldsymbol{e}, f)$ be a pair generated in the GVW algorithm, where $x^\alpha \boldsymbol{e}$ is the signature of this pair. Besides storing $x^\alpha \boldsymbol{e}$ and f in our algorithm, we also generate and record a set of polynomials $A = \{a_1, \cdots, a_k\}$ such that $\mathrm{lm}(a_i) = \mathrm{lm}(f)$, a_i is equivalent to f in some senses and a_i is much sparser than f where $i = 1, 2, \cdots, k$. Before eliminating matrices, we substitute the polynomial tf with some ta_i in a particular way to get sparser matrices, where t is a monomial. Since the matrices get sparser after substitutions, they can be eliminated more efficiently. Hence, the GVW algorithm can be speeded up. We give detailed proofs for this substituting method.

We present two specific algorithms, named as Block-GVW and LM-GVW. These two algorithms use the substituting method in different ways, and the combination of them is the Sub-GVW algorithm.

We have implemented the Sub-GVW algorithm over Boolean polynomial rings. The routine `gvw_ple()`[22] modified from `mzd_ple()`[23] is used for eliminating dense matrices. The sparse

elimination routines are implemented on our own. We test many Boolean polynomial systems with the Sub-GVW algorithm, and compare the sparsity and eliminating time of the matrices before and after the substitutions. Experimental results show our new substitution method does help generate sparser matrices and hence improve the efficiency of the algorithm.

This paper is organized as follows. We illustrate our main ideas in Section 2. Next, we present the Sub-GVW algorithm in Section 3. Experimental results are provided in Section 4. Concluding remarks follow in Section 5.

2 Main Ideas

As is known, almost all the computing time of a Gröbner basis is spent on polynomial reductions. Linear algebraic techniques have been proven very effective on speeding up the implementations of Gröbner basis algorithms. Thus, we only consider the implementation of GVW using linear algebra.

Problem description Generally, in order to use sophistical linear algebraic techniques, we convert polynomials to rows of matrices, and hence, reductions of polynomials are done by eliminating corresponding matrices. In signature-based Gröbner basis algorithms, each row is designated with a signature which is comparable. Since the correctness of criteria used in signature-based algorithms should be ensured, polynomials or rows with larger signatures can only be reduced by polynomials or rows with smaller signatures. This leads to a one-direction reduction of the matrices. Specifically, we can sort the rows in a matrix by an ascending order on their signatures, with the top row having the smallest signature. Then the matrix can be only eliminated in a top-down manner, i.e., bottom rows can only be eliminated by top rows and the reverse elimination is forbidden. Thus, rows of the matrices cannot be freely swapped during the elimination. In this paper, we are going to present an efficient method for doing one-direction reductions.

In [16], Faugère and Lachartre presented an efficient method for eliminating matrices during Gröbner basis computations. For simplification, we call this method as **Faugère's method** in the rest of this paper. This method is illustrated in Figure 1. Specifically, since matrices appearing in Gröbner basis are always in a quasi-triangular form, by doing appropriate row and column swaps, one can transform the matrix to the form (a), where A is an upper triangular matrix, and B, C, D are rectangular matrices. To eliminate the matrix to echelon form, we can compute $A^{-1} \cdot B$, $C \cdot (A^{-1}B)$, and $D + CA^{-1}B$, and these procedures are shown in (b), (c), (d) respectively. At last, the matrix $D + CA^{-1}B$ is eliminated to D' in (e). Generally, the matrix A and C are quite sparse, so the computation $A^{-1} \cdot B$ and $C \cdot (A^{-1}B)$ can be done very efficiently using sparse linear algebraic techniques. The matrix $D + CA^{-1}B$ usually has a density of 50%, so this elimination is usually done by dense matrix operations. In many of our experiments, the number of rows in $D + CA^{-1}B$ is only 1/10 of the rows of A, so most of the computing time is spent on sparse eliminations.

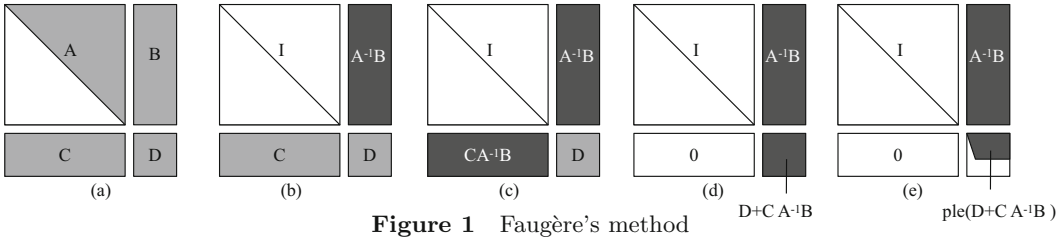

Figure 1 Faugère's method

A prerequisite of using the above method is that the rows of matrices can be freely swapped. This can be easily done in the F4 algorithm. To exploit this technique in signature-based algorithms, a natural way is to split the matrix into blocks. Precisely, the rows of the matrix are divided into several blocks, and the signatures of the rows in the upper blocks are all smaller than the signatures of those in the bottom blocks. For a simple example, we consider two blocks as shown in Figure 2. To eliminate the matrix, we first reduce the top block by one-direction elimination and get (b), which is not a reduced echelon form. Useful polynomials should be collected from this non-reduced form of Block I at this time. Before eliminating the bottom block, we rearrange the rows and columns of the reduced polynomials obtained in the top block, and then convert the matrix to the form in (c). Note that this rearrangement of rows does not affect the eliminations that will be done in Block II, because all rows in Block I have smaller signatures than those in Block II. Next, Faugère's method is used to eliminate the bottom block. Note that the last dense matrix in Faugère's method should be eliminated in one-direction, so the technique in [22] can be used.

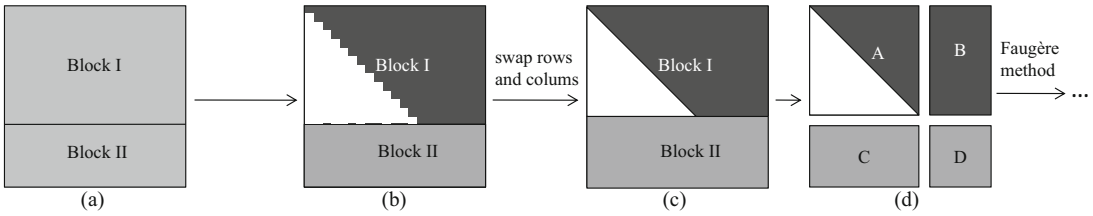

Figure 2 Tradition method of elimination by blocks

There is one flaw about the above method. That is, the upper block usually becomes denser after the one-direction elimination, then in Faugère's method (Figure 1) converting the matrix from the form (b) to the form (c) will cost more time. The problem becomes severe when the number of rows grows larger, which usually happens when the matrix is divided into several blocks.

To resolve this flaw, we propose our first method.

Method I The basic idea is to substitute rows in Block I of Figure 2 (c) to sparser ones, and the method is illustrated in Figure 3. There are two kinds of polynomials/rows in the echelon form of Block I in Figure 2 (c). The first kind of polynomials do not have new leading monomials compared with polynomials in Block I of Figure 2 (a), and the other kind

has new leading monomials. For simplicity, we assume the polynomials/rows with new leading monomials are at the bottom part of Block I, e.g., the polynomial denoted as g in Figure 3 (a).

In Method I, we leave the second kind of polynomials (e.g., g) unsubstituted. We only consider the polynomials without new leading monomials. Take the polynomial f in Figure 3 (a) for example, we record a polynomial p in previous computations. This polynomial p is obtained from *previous matrices* and is sparser than f. Besides, we also need $\mathrm{lm}(p) = \mathrm{lm}(f)$ and the difference $p - f$ lies in the linear space generated by the rows/polynomials in Block I, by which we mean p is equivalent to f. After doing substitutions like $f \longrightarrow p$, we get the matrix (b). The substituted polynomials are then converted to the upper matrices in (c), and the polynomials with new leading monomials are generally very dense and are included in the right bottom matrix. At last, Faugère's method is used to eliminate the bottom matrices in Figure 3 (c). The algorithm using Method I is named as the Block-GVW algorithm, which is detailed in Subsection 3.3.

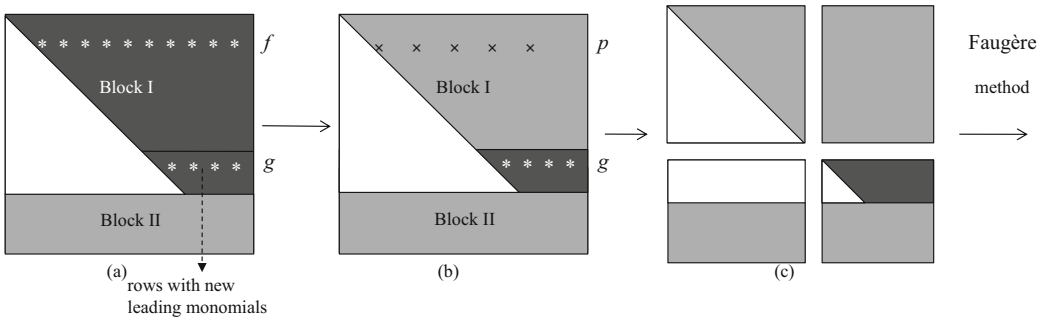

Figure 3 The Block-GVW algorithm

Remark The polynomial p, which is used to substitute others, often comes from previous lower degree matrices. The exact signature of p is usually unknown, because p is often generated by using a double-direction elimination.

Method II Since substituting polynomials leads to sparser matrices and better elimination efficiencies, it is natural to substitute as more polynomials as possible. This natural idea results in our second method illustrated in Figure 4. Firstly, we construct all polynomials with signatures in (a). Next, we swap rows without considerations of signature orderings and put all J-pairs to be reduced at the bottom of the matrix, and the polynomials used to reduced others are at the top in (b). By doing substitutions for the polynomials in the upper part, we get the matrix (c). At last, the whole matrix is split into four sub-matrices, and Faugère method is used for elimination.

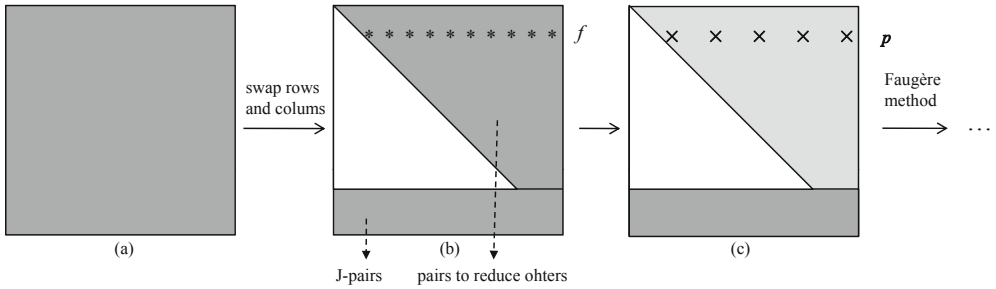

Figure 4 The LM-GVW algorithm

The above elimination is very efficient, because the matrix after substitution is sparser and the whole elimination is not done in one-direction. Since we rearrange the polynomials in (b) without considerations of signatures, we cannot find the correct signatures for the final eliminating results. But it is easy to prove that the new leading monomials of the results are contained in those obtained by using one-direction elimination to the matrix in (a). This implies, before doing one-direction elimination to the whole matrix, we can obtain the new leading monomial information efficiently by Method II. So Method II is very useful for quickly finding linear or mutant polynomials, which can help speed up the algorithm significantly. Method II is introduced as the LM-GVW algorithm in Subsection 3.4, where "LM" shorts for leading monomial.

3 Speeding up the GVW Algorithm via a Substituting Method

This section is organized as follows. We revisit the GVW algorithm and rewrite it as a matrix style in the first two subsections. We present the Block-GVW algorithm and LM-GVW algorithm in Subsections 3.3 and 3.4, in which we show how our substituting methods are used. By combining Block-GVW and LM-GVW, we obtain the Sub-GVW algorithm. We give a toy example to illustrate the Sub-GVW algorithm in the last subsection.

3.1 The GVW Algorithm Revisited

In this part, some necessary notations are given and then the GVW algorithm is introduced briefly. For more details, please refer to [12].

Let $R := K[x_1, \cdots, x_n]$ be a polynomial ring over a field K with n variables, and $\{f_1, \cdots, f_m\}$ is a finite subset of R. Consider an ideal I:

$$I = \langle f_1, \cdots, f_m \rangle = \{p_1 f_1 + \cdots + p_m f_m \mid p_1, \cdots, p_m \in R\}.$$

Given some monomial ordering on R, we want to compute a Gröbner basis for I.

Let $\mathbb{F}_2 := (f_1, \cdots, f_m) \in R^m$, and consider the following R-module of $R^m \times R$:

$$\boldsymbol{M} = \{(\boldsymbol{u}, f) \in R^m \times R \mid \boldsymbol{u} \cdot \mathbb{F}_2 = f\}.$$

Then the R-module $\boldsymbol{M}$ is generated by $\{(\boldsymbol{e}_1, f_1), \cdots, (\boldsymbol{e}_m, f_m)\}$, where $\boldsymbol{e}_i$ is the i-th unit vector of R^m.

A monomial in R has the form $x^\alpha = \Pi_{i=1}^n x_i^{a_i}$, where $\alpha = (a_1, \cdots, a_n) \in \mathbb{N}^n$ and $\mathbb{N}$ is the set of all non-negative integers. A monomial in R^m is of the form $x^\alpha \boldsymbol{e}_i$, where $1 \leq i \leq m$ and $\alpha \in \mathbb{N}^n$. For monomials in R^m, if $\boldsymbol{e}_i = \boldsymbol{e}_j$ and x^α divides x^β, we say that $x^\alpha \boldsymbol{e}_i$ divides $x^\beta \boldsymbol{e}_j$ (or $x^\alpha \boldsymbol{e}_i \mid x^\beta \boldsymbol{e}_j$ for short). Furthermore, we define the quotient $(x^\beta \boldsymbol{e}_i)/(x^\alpha \boldsymbol{e}_i) = x^{\beta-\alpha} \in R$.

Fix any monomial ordering $\prec_p$ on R and any monomial ordering $\prec_s$ on R^m (subscripts p and s stand for polynomial and signature respectively). Please note that $\prec_s$ may or may not be related to $\prec_p$ in theory, although we always assume $\prec_s$ is **compatible** with $\prec_p$ practically, i.e., $x^\alpha \prec_p x^\beta$ if and only if $x^\alpha \boldsymbol{e}_i \prec_s x^\beta \boldsymbol{e}_i$ for $1 \leq i \leq m$. For the sake of simpleness, we use the following notations for leading monomials:

$$\mathrm{lm}(f) = \mathrm{lm}_{\prec_p}(f) \text{ and } \mathrm{lm}(\boldsymbol{u}) = \mathrm{lm}_{\prec_s}(\boldsymbol{u}),$$

for any $f \in R$ and any $\boldsymbol{u} \in R^m$. Leading monomials of $f \in R$ and $\boldsymbol{u} \in R^m$ are monomials without coefficients in R and R^m respectively. We define $\mathrm{lm}(f) = 0$ if $f = 0$, and $0 \prec_p x^\alpha$ for any non-zero monomial x^α in R; similarly for monomials in R^m. In the rest of this paper, we use $\prec$ instead of $\prec_p$ and $\prec_s$ if no confusion occurs.

We should emphasize that we always assume $\prec_p$ is a graded monomial ordering and $\prec_s$ is compatible with $\prec_p$ in this paper.

The **signature** of $(\boldsymbol{u}, f)$ is defined as $\mathrm{lm}(\boldsymbol{u})$. If $\mathrm{lm}(\boldsymbol{u}) = x^\alpha \boldsymbol{e}_i$ is the signature of $(\boldsymbol{u}, f)$, we call i as the **index** of $(\boldsymbol{u}, f)$. This definition of signature is the same as used in GVW, but not in [6, 9]. The differences are discussed in [12].

Let $(\boldsymbol{u}, f) \in \boldsymbol{M}$ and $B \subset \boldsymbol{M}$. We give the concept of **top-reducible**. We say $(\boldsymbol{u}, f)$ is **top-reducible** by B, if there exists $(\boldsymbol{v}, g) \in B$ with $g \neq 0$, such that $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ and $\mathrm{lm}(\boldsymbol{u}) \succeq \mathrm{lm}(t\boldsymbol{v})$ where $t = \mathrm{lm}(f)/\mathrm{lm}(g)$. The corresponding **top-reduction** is then

$$(\boldsymbol{u}, f) - ct(\boldsymbol{v}, g) = (\boldsymbol{u} - ct\boldsymbol{v}, f - ctg),$$

where $c = \mathrm{lc}(f)/\mathrm{lc}(g)$, and $\mathrm{lc}(f)$, $\mathrm{lc}(g)$ are the leading coefficients of f and g. We call the top-reduction **regular**, if $\mathrm{lm}(\boldsymbol{u}) \succ \mathrm{lm}(t\boldsymbol{v})$; and **super** if $\mathrm{lm}(\boldsymbol{u}) = \mathrm{lm}(t\boldsymbol{v})$.[†] Clearly, $(\boldsymbol{u} - ct\boldsymbol{v}, f - ctg)$ is also an element in $\boldsymbol{M}$.

A subset G of $\boldsymbol{M}$ is called a **strong Gröbner basis** for $\boldsymbol{M}$ if every nonzero pair (pairs $\neq (\boldsymbol{0}, 0)$) in $\boldsymbol{M}$ is top-reducible by G. According to Proposition 2.2 of [12], let $G := \{(\boldsymbol{v}_i, g_i) \mid 1 \leq i \leq s\}$ be a strong Gröbner basis for $\boldsymbol{M}$. Then $\{g_i \mid 1 \leq i \leq s\}$ is a Gröbner basis for $I = \langle f_1, \cdots, f_m \rangle$.

Joint pairs/J-pairs are used to compute strong Gröbner bases. Suppose $(\boldsymbol{u}, f), (\boldsymbol{v}, g) \in \boldsymbol{M}$ are two pairs with f and g both nonzero. Let $t = \mathrm{lcm}(\mathrm{lm}(f), \mathrm{lm}(g))$, $t_f = t/\mathrm{lm}(f)$ and $t_g = t/\mathrm{lm}(g)$. Then the **J-pair** of $(\boldsymbol{u}, f)$ and $(\boldsymbol{v}, g)$ is defined as: $t_f(\boldsymbol{u}, f)$ (or $t_g(\boldsymbol{v}, g)$), if $\mathrm{lm}(t_f \boldsymbol{u}) \succ \mathrm{lm}(t_g \boldsymbol{v})$ (or $\mathrm{lm}(t_f \boldsymbol{u}) \prec \mathrm{lm}(t_g \boldsymbol{v})$). For the case $\mathrm{lm}(t_f \boldsymbol{u}) = \mathrm{lm}(t_g \boldsymbol{v})$, the J-pair is not defined. We note that the J-pair of $(\boldsymbol{u}, f), (\boldsymbol{v}, g) \in \boldsymbol{M}$ is also a pair in $\boldsymbol{M}$. Let $t_f(\boldsymbol{u}, f)$ be the J-pair of $(\boldsymbol{u}, f)$ and $(\boldsymbol{v}, g)$, the **degree** of $t_f(\boldsymbol{u}, f)$ is defined as $\deg(t_f f)$, i.e., the degree of the polynomial part. We call a J-pair is of $G \subset \boldsymbol{M}$ for short, if it is the J-pair of two pairs in G.

[†]Regular top-reduction defined here is slightly different from its original version in[12], but this will not affect proofs of related propositions and theorems.

For a pair $(\boldsymbol{u}, f) \in \boldsymbol{M}$ and a set $G \subset \boldsymbol{M}$, we call $(\boldsymbol{u}, f)$ is **covered** by G, if there is a pair $(\boldsymbol{v}, g) \in G$, such that $\mathrm{lm}(\boldsymbol{v})$ divides $\mathrm{lm}(\boldsymbol{u})$ and $t\mathrm{lm}(g) \prec \mathrm{lm}(f)$ (strictly smaller) where $t = \mathrm{lm}(\boldsymbol{u})/\mathrm{lm}(\boldsymbol{v})$. For convenience, we say a pair $(\boldsymbol{w}, h) \in \boldsymbol{M}$ is a **syzygy** pair if $h = 0$.

Gao, et al. provided a simple characterization of strong Gröbner bases.

Theorem 3.1 (see [12]) *Suppose $G \subset \boldsymbol{M}$ contains pairs with signatures $\{\boldsymbol{e}_1, \cdots, \boldsymbol{e}_m\}$. Then G is a strong Gröbner basis for $\boldsymbol{M}$ if and only if every J-pair of G is covered by G.*

It should be noted that checking whether a pair is covered by G does not need any reduction of polynomials. This deduces the various rewriting rules used in the literature. The following two criteria are rephrased from the rewriting rules.

[Syzygy Criterion] For a J-pair $t_f(\boldsymbol{u}, f)$ of a set $G \in \boldsymbol{M}$, if there exists a syzygy pair $(\boldsymbol{v}, 0) \in G$ such that $\mathrm{lm}(\boldsymbol{v})$ divides $t_f\mathrm{lm}(\boldsymbol{u})$, then this J-pair can be discarded.

[Second Criterion] For a J-pair of a set $G \in \boldsymbol{M}$, if this J-pair is covered by G, then this J-pair can be discarded.

In this paper, we call the second criterion **Rewriting Criterion**. In [9], Arri and Perry proposed a quite similar criterion to Rewriting Criterion. For more comments on Arri-Perry's criterion and Rewriting Criterion, please see [12, 20].

We modify the original GVW slightly and get the following algorithm. The output of a Gröbner basis for the syzygy module of input polynomials is deleted, because we only care about the Gröbner basis of input polynomials in this paper.

Algorithm 1: The GVW algorithm

Input : $f_1, \cdots, f_m \in R = K[x_1, \cdots, x_n]$; $\prec_p$ and $\prec_s$ monomial orderings for R and R^m.

Output: A Gröbner basis of $I = \langle f_1, \cdots, f_m \rangle$.

```
1  begin
2      G⟵{(e_i, f_i) | 1 ≤ i ≤ m}
3      H⟵{(f_j e_i − f_i e_j, 0) | 1 ≤ i < j ≤ m}
4      JP⟵all J-pairs of G
5      while JP ≠ ∅ do
6          Let t(u, f) ∈ JP and remove t(u, f) from JP.
7          if t(u, f) is rejected by either Syzygy or Rewriting Criterion then
8              GotoLine 5
9          (w, h)⟵Regular top-reduce t(u, f) by G.
10         if h = 0 then
11             H⟵H ∪ {(w, 0)}
12         else
13             JP⟵JP ∪ {J-pair of (w, h) and (v, g) | (v, g) ∈ G}
14             G⟵G ∪ {(w, h)}
15     return {g | (v, g) ∈ G}
```

Some remarks on the GVW algorithm are listed in the following.

1) At Step 6, a J-pair can be selected from the set JP in any order. We always select the J-pairs with the minimal degree in the algorithms of this paper.

2) In [12], Proposition 2.2 guarantees the correctness of GVW when J-pairs are computed in any order, and the finite termination of GVW is proved by Theorem 3.1 when monomial orderings of R and R^m are compatible.

3) The GVW algorithm in [12] retains only one J-pair (the one with the minimal polynomial part) when there are several J-pairs having the same signature. This process can be implied by the "cover check" at Step 7.

4) It is emphasized that for a pair $(\boldsymbol{u}, f) \in \boldsymbol{M}$, only $(\mathrm{lm}(\boldsymbol{u}), f)$ is stored during the implementation of the GVW algorithm. Other related conceptions, such as top-reduction, J-pairs and cover, are defined similarly. More details could be found in [12].

3.2 The Matrix-GVW Algorithm

In this section, we rewrite the GVW algorithm in a matrix style and give the Matrix-GVW algorithm. Instead of reducing J-pairs one by one, Matrix-GVW uses batch processing to reduce J-pairs. This batch processing can be implemented through eliminating matrices, which could significantly speed up the implementation of GVW.

The Matrix-GVW algorithm is given below.

Algorithm 2: The Matrix-GVW algorithm

Input : $f_1, \cdots, f_m \in R = K[x_1, \cdots, x_n]$; $\prec_p$ and $\prec_s$, monomial orderings on R and R^m.
Output: A Gröbner basis of $I = \langle f_1, \cdots, f_m \rangle$.

1 **begin**
2 $G \longleftarrow \{(\boldsymbol{e}_i, f_i) \mid 1 \le i \le m\}$
3 $H \longleftarrow \{(f_j \boldsymbol{e}_i - f_i \boldsymbol{e}_j, 0) \mid 1 \le i < j \le m\}$
4 $\mathsf{JP} \longleftarrow$ all the J-pairs of G
5 **while** $\mathsf{JP} \neq \emptyset$ **do**
6 $d \longleftarrow \min\{\deg(\mathrm{lm}(tf)) \mid t(\boldsymbol{u}, f) \in \mathsf{JP}\}$
7 $\mathsf{minJP} \longleftarrow \{t(\boldsymbol{u}, f) \mid t(\boldsymbol{u}, f) \in \mathsf{JP}, \deg(\mathrm{lm}(tf)) = d\}$
8 $\mathsf{JP} \longleftarrow \mathsf{JP} \setminus \mathsf{minJP}$
9 $\mathsf{minJP} \longleftarrow \mathsf{minJP} \setminus \{$J-pairs rejected by Syzygy and Rewriting Criterion$\}$
10 $M \longleftarrow \mathtt{SymbolicProcess}(\mathsf{minJP}, G, \prec_p, \prec_s)$
11 $\widetilde{M} \longleftarrow \mathtt{OneDirectionElimination}(M, \prec_s)$
12 $G, \mathsf{JP}, H \longleftarrow \mathtt{Update}(\widetilde{M}, G, \mathsf{JP}, H, \prec_s)$
13 **return** $\{g \mid (\boldsymbol{v}, g) \in G\}$

The input of Matrix-GVW is a set of polynomials and the output is a Gröbner basis of the input system. Firstly, let G be the set of initial pairs, and all the J-pairs of G are generated in Step 4. In every loop, Matrix-GVW chooses the J-pairs from the set JP with the minimal degree and rejects some J-pairs by Syzygy and Rewriting Criterion in Steps 6–9. Then the

algorithm builds the set $M \subset \boldsymbol{M}$ in Step 10, and M consists of J-pairs to be reduced and pairs that are used to reduce others. Reductions are done by batch processing in Step 11, which are usually processed via eliminations of matrices. At last, in Step 12 the set of G, JP and H are updated by $\widetilde{M}$.

The function `SymbolicProcess(·)` returns a set $M \subset \boldsymbol{M}$ which contains all pairs needed for reductions, including J-pairs to be reduced and pairs that are used to reduce others. Given the J-pairs to be reduced, the set `Todo` contains all the monomials not bigger than the greatest monomial in JP. The steps from Step 6 to Step 7 find out the polynomials whose leading monomials are in `Todo`. Please note that, when there are more than one polynomials having the same leading monomial, we always choose the one with the minimal signature in Steps 8–9. We emphasize this step is very important, since it ensures all J-pairs in M can be fully reduced.

Function SymbolicProcess

Input : JP, a set of J-pairs; G, a set of pairs in $\boldsymbol{M}$; $\prec_p$, $\prec_s$, monomial orderings on R, R^m.

Output: M, a set of pairs in $\boldsymbol{M}$.

1 **begin**
2 $D \longleftarrow \emptyset$
3 Todo$\longleftarrow$ all monomials not bigger than $\max\{\mathrm{lm}(tf) \mid t(\boldsymbol{u}, f) \in \mathsf{JP}\}$
4 **while** Todo $\neq \emptyset$ **do**
5 Pick up a monomial $m \in$ Todo, and remove m from Todo
6 **if** $\exists(\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g) \mid m$ **then**
7 $G_0 \longleftarrow \{t(\boldsymbol{v}, g) \mid \mathrm{lm}(g)\,\mathrm{divides}\, m, t = m/\mathrm{lm}(g), (\boldsymbol{v}, g) \in G\}$
8 find $t_0(\boldsymbol{v}_0, g_0) \in G_0$ with the minimal signature w.r.t. $\prec_s$
9 $D = D \cup \{t_0(\boldsymbol{v}_0, g_0)\}$
10 $M \longleftarrow \mathsf{JP} \cup D$
11 **return** M

Function `OneDirectionElimination(·)` reduces pairs in M. It is a one-direction reduction, because it only allows pairs with smaller signatures to reduce pairs with bigger ones. In each loop, the function always chooses the pair $(\boldsymbol{u}, f)$ with the smallest signature in `Todo` to reduce in Step 5. For candidates to reduce $(\boldsymbol{u}, f)$, we only consider pairs in $\widetilde{M}$ whose signatures are strictly smaller than $(\boldsymbol{u}, f)$. Whenever $(\boldsymbol{u}, f)$ is not reducible, it is appended to $\widetilde{M}$. This function terminates when all pairs in `Todo` are reduced. Please remark that, due to Steps 6–9 in Function `SymbolicProcess(·)`, every pair in $\widetilde{M}$ cannot be further regular-top reduced by G.

Function OneDirectionElimination

Input : M, a set of pairs in $\boldsymbol{M}$; $\prec_s$, a monomial ordering on R^m.
Output: $\widetilde{M}$, a set of pairs in $\boldsymbol{M}$.

```
1  begin
2      Todo⟵M
3      M̃⟵∅
4      while Todo ≠ ∅ do
5          Pick up (u, f) having the smallest signature in Todo and remove it from Todo
6          if ∃(v, g) ∈ M̃ s.t. lm(g) = lm(f) ≠ 0 and lm(v) ≺_s lm(u) then
7              Todo⟵Todo ∪ {(u, f) − (v, g)}
8          else
9              M̃⟵M̃ ∪ {(u, f)}
10     return M̃
```

In Matrix-GVW, Function `OneDirectionElimination(·)` is implemented in a matrix style. First, all polynomials in M are converted into rows of a matrix, i.e., for any $(\boldsymbol{u}, f) \in M$, we write the coefficients of f as entries of a row and we say $\mathrm{lm}(\boldsymbol{u})$ is the signature of this row. Second, we sort the rows of the matrix by an ascending order on their signatures. Specifically, the row with the smallest signature is on the top of the matrix. Next, the matrix is eliminated in one direction. That is, a row can only be eliminated by rows above it with smaller signatures. Finally, when the matrix is eliminated, all rows are transformed back to pairs and stored in $\widetilde{M}$. Please note that, during the practical implementation only $(\mathrm{lm}(\boldsymbol{u}), f)$ is stored instead of $(\boldsymbol{u}, f)$, which is the same as done in [12].

The function `Update(·)` collects new pairs in $\widetilde{M}$ and generates new J-pairs. We define two kinds of new pairs in $\widetilde{M}$ according to their signatures and leading monomials. By saying the leading monomial of a pair $(\boldsymbol{u}, f)$, we mean $\mathrm{lm}(f)$ in the rest of this section.

1) For $(\boldsymbol{u}, f) \in \widetilde{M}$, if $\mathrm{lm}(f)$ cannot be generated by the leading monomials of pairs in G, i.e., there is no leading monomials of pairs in G dividing $\mathrm{lm}(f)$, then $(\boldsymbol{u}, f)$ is new.

2) For $(\boldsymbol{u}, f)$ such that $\mathrm{lm}(f)$ can be generated by the leading monomials of pairs in G, we check whether $(\boldsymbol{u}, f)$ has a relative smaller signature. That is, if there exists $(\boldsymbol{v}, g) \in G$ such that $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ and $\mathrm{lm}(t\boldsymbol{v}) \preceq_s \mathrm{lm}(\boldsymbol{u})$ where $t = \mathrm{lm}(f)/\mathrm{lm}(g)$, then we say $(\boldsymbol{u}, f)$ is not a new pair; otherwise, $(\boldsymbol{u}, f)$ is new.

In Function `Update(·)`, the above two kinds of new pairs are collected in M_1^+ and M_2^+ respectively. Next, all the new pairs in M_1^+ and M_2^+ are appended to G and the J-pairs are updated correspondingly in Steps 8–10. The updated sets of G and `JP` are returned at last.

Function Update

Input : $\widetilde{M}$, a set of pairs in $\boldsymbol{M}$ returned by Function `OneDirectionElimination(·)`; G, a set of pairs in $\boldsymbol{M}$; JP, a set of J-pairs; H, a set of syzygy pairs in $\boldsymbol{M}$, i.e., $\forall(\boldsymbol{w}, h) \in H$ we have $h = 0$; $\prec_s$, a monomial ordering on R^m.

Output: The updated sets of G, JP and H.

begin
  $(M_1^+, M_2^+) \longleftarrow (\emptyset, \emptyset)$
  for every $(\boldsymbol{u}, f) \in \widetilde{M}$ with $f \neq 0$ **do**
    if $\nexists(\boldsymbol{v}, g) \in G\, s.t.\, \mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ **then**
      $\boldsymbol{M}_1^+ \longleftarrow \boldsymbol{M}_1^+ \cup \{(\boldsymbol{u}, f)\}$
    else if $\frac{\mathrm{lm}(f)}{\mathrm{lm}(g)}\mathrm{lm}(\boldsymbol{v}) \succ_s \mathrm{lm}(\boldsymbol{u})$ for $\forall(\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g)|\mathrm{lm}(f)$ **then**
      $\boldsymbol{M}_2^+ \longleftarrow \boldsymbol{M}_2^+ \cup \{(\boldsymbol{u}, f)\}$
  for every $(\boldsymbol{u}, f) \in M_1^+ \cup M_2^+$ **do**
    JP$\longleftarrow$JP $\cup$ {J-pairs of $(\boldsymbol{u}, f)$ and $(\boldsymbol{v}, g) \mid (\boldsymbol{v}, g) \in G$}
    $G \longleftarrow G \cup \{(\boldsymbol{u}, f)\}$
  $H \longleftarrow H \cup \{(\boldsymbol{w}, 0) \mid (\boldsymbol{w}, 0) \in \widetilde{M}\}$
  return G, JP, H

We prove the termination and correctness of the Matrix-GVW algorithm below.

Theorem 3.2 *The Matrix-GVW algorithm terminates in finite steps.*

Proof To prove the theorem, we consider a map $\sigma : R^m \times R \to k[Y, Z, W]$, where $Y = \{y_1, \cdots, y_n\}$, $Z = \{z_1, \cdots, z_m\}$, and $W = \{w_1, \ldots, w_n\}$ are new variables. For any $(\boldsymbol{u}, f) \in \boldsymbol{M}$ with $\mathrm{lm}(\boldsymbol{u}) = x_1^{a_1} \cdots x_n^{a_n} \boldsymbol{e}_i \neq 0$ and $\mathrm{lm}(f) = x_1^{b_1} \cdots x_n^{b_n} \neq 0$, we define

$$\sigma(\boldsymbol{u}, f) = y_1^{a_1} \cdots y_n^{a_n} z_i w_1^{b_1} \cdots w_n^{b_n} \in k[Y, Z, W].$$

For a set $F = \{(\boldsymbol{u}, f) \mid f \neq 0, (\boldsymbol{u}, f) \in \boldsymbol{M}\}$, we define $\sigma(F) = \{\sigma(\boldsymbol{u}, f) \mid (\boldsymbol{u}, f) \in F\}$.

Let G_{n-1} and H_{n-1} denote the sets G and H respectively before the n-th "while" loop starts and G_n, H_n be the sets G, H after the n-th loop finishes. In each "while" loop, one of the following three cases must happen.

(a) Assume all the J-pairs are rejected by the criteria, then no new pairs are added to G. So the number of J-pairs in JP decreases strictly.

(b) Assume Case (a) does not hold and there are some J-pairs reduced to 0. Then they are appended to the set H_n. Since all the J-pairs covered by syzygies have been discarded, the signatures of new syzygies are not divisible by the signatures of pairs in H_{n-1}. Thus, we must have $\langle H_{n-1}\rangle \subsetneq \langle H_n\rangle$.

(c) Assume Cases (a) and (b) do not hold and all J-pairs are reduced to non-syzygy pairs. Then we claim $\langle\sigma(G_{n-1})\rangle \subsetneq \langle\sigma(G_n)\rangle$. On one hand, if the first kind of new pairs are generated, then the claims hold directly by the definition of σ. On the other hand, we assume only the second kind of new pairs are generated. Let $(\boldsymbol{w}, h)$ be a new pair of the second kind. If there

exists $(\boldsymbol{v}, g) \in G_{n-1}$ such that $\sigma(\boldsymbol{v}, g)$ divides $\sigma(\boldsymbol{w}, h)$, then we must have $\mathrm{lm}(\boldsymbol{v})$ divides $\mathrm{lm}(\boldsymbol{u})$ and $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ by definition of σ. Denote $s = \mathrm{lm}(\boldsymbol{u})/\mathrm{lm}(\boldsymbol{v})$ and $t = \mathrm{lm}(f)/\mathrm{lm}(g)$. If $s \succeq t$, then $(\boldsymbol{w}, h)$ has not been fully reduced, and this is a contradiction due to Steps 8–9 in Function `SymbolicProcess(·)`; otherwise, $(\boldsymbol{w}, h)$ is covered by $(\boldsymbol{v}, g)$ and should have been discarded by criteria. In all, we have that none of pairs in G_{n-1} can divide $(\boldsymbol{w}, h)$, and the claim is proved.

Since the polynomial ring over a field is Noetherian. None of the above cases can happen infinite times. Therefore, Matrix-GVW algorithm terminates in finite steps. ∎

Theorem 3.3 *The Matrix-GVW algorithm computes a Gröbner basis for the input system.*

Proof We want to prove this proposition via Theorem 3.1. That is, if all the J-pairs of G are covered by G, then G is a strong Gröbner basis. Note that the set G in Theorem 3.1 is splited into sets G and H in Matrix-GVW.

The J-pairs rejected by the criteria are obviously covered by $G \cup H$. Then we consider each J-pair $(\boldsymbol{u}, f)$ which is not rejected. Then $(\boldsymbol{u}, f)$ is reduced to either $(\boldsymbol{u}, 0)$ or $(\boldsymbol{u}, g)$ where $\mathrm{lm}(g) \prec_p \mathrm{lm}(f)$, and $(\boldsymbol{u}, 0)$ or $(\boldsymbol{u}, g)$ will be appended to H or G, respectively. Hence, $(\boldsymbol{u}, f)$ will be covered by this new H or G.

So the Matrix-GVW algorithm terminates only when all J-pairs of G are covered by $G \cup H$. Matrix-GVW computes a Gröbner basis according to Theorem 3.1. ∎

Matrix-GVW could accelerate GVW significantly, since it takes advantage of the batch processing for J-pairs reductions. However, compared with batch processing used in F4, Matrix-GVW still has two restrictions. First, the elimination of matrices must be done from one direction. Matrix-GVW only allows rows with lower signatures to reduce those with higher signatures. Thus, we cannot swap rows as our wishes during the elimination. And hence, the technique introduced in [16] cannot be used directly. Second, polynomials in Matrix-GVW are generally denser than those in F4, because we can only obtain the echelon form of matrix but not reduced echelon form as done in F4. This may lead to denser matrices and slow down the eliminations.

On seeing these restrictions, we propose the Substituting GVW algorithm, or Sub-GVW for short. The Sub-GVW algorithm is an integration of the Block-GVW algorithm and the LM-GVW algorithm. We introduce these two algorithms in the following subsections.

3.3 The Block-GVW Algorithm

The Block-GVW algorithm improves the Matrix-GVW algorithm by dividing the whole matrix into several blocks. When eliminating each block, sparser equivalent polynomials are used to substitute denser polynomials, which decreases the density of the matrix and obtains a better eliminating efficiency. Faugère's method[16] is also used. The main idea is illustrated in Figure 3 of Section 2.

Generally, we can divide the matrix into k blocks. For sake of simplicity, we only discuss the Block-GVW algorithm for $k = 2$, named as 2-Block-GVW algorithm. The algorithm can

be extended to $k = 3, 4, \cdots$ naturally.

We introduce some necessary notations in the followings.

Let $\{f_1, \cdots, f_m\} \subset R = K[x_1, \cdots, x_n]$ be m polynomials, $\prec_p$ be a graded monomial ordering and $\prec_s$ be an position over term extension of $\prec_p$ such that $\boldsymbol{e}_1 \prec_s \boldsymbol{e}_2 \prec_s \cdots \prec_s \boldsymbol{e}_m$.

To do substitutions, we need to store more polynomials inside the pairs. We consider the 3-tuple $(\boldsymbol{u}, f, p)$, where $(\boldsymbol{u}, f) \in \boldsymbol{M}$ and $p \in R$. The polynomial p is used to substitute f, and it is only an auxiliary polynomial. Initially, we set $p := f$, and update p in Function `UpdateBlock(·)`. Concepts defined for $(\boldsymbol{u}, f)$ can be extended to $(\boldsymbol{u}, f, p)$ by ignoring p. These concepts include Syzygy Criterion, Rewriting Criterion, and J-pairs. But we do not consider the operations of the 3-tuples, such as additions or multiplications by monomials.

Please remark that, we only append one polynomial p to the pair $(\boldsymbol{u}, f)$ when the block number k is 2. For $k > 2$, we need to append $k - 1$ polynomials correspondingly.

Given a matrix M corresponding to the set $\{(\boldsymbol{u}, f, p)\}$ in the 2-Block-GVW algorithm, we divide M to M_I and M_{II} by a fixed $\boldsymbol{e}_l \in R^m$ where $1 < l \leq m$. That is, $M_I = \{t(\boldsymbol{u}, f, p) \in M \mid \mathrm{lm}(\boldsymbol{u}) \prec_s \boldsymbol{e}_l\}$ and $M_{II} = M \setminus M_I$.

The 2-Block-GVW algorithm is described below.

Algorithm 3: The 2-Block-GVW algorithm

Input : $f_1, \cdots, f_m \in R$; $\prec_p$ and $\prec_s$, monomial orderings on R and R^m.

Output: A Gröbner basis of $I = \langle f_1, \cdots, f_m \rangle$.

begin
  $G \longleftarrow \{(\boldsymbol{e}_i, f_i, f_i) \mid 1 \leq i \leq m\}$
  $H \longleftarrow \{(\boldsymbol{e}_i f_j - \boldsymbol{e}_j f_i, 0) \mid 1 \leq i < j \leq m\}$
  $\mathsf{JP} \longleftarrow$ all the J-pairs of G
  while $\mathsf{JP} \neq \emptyset$ **do**
    $d \longleftarrow \min\{\deg(t\mathrm{lm}(f)) \mid (t(\boldsymbol{u}, f, p) \in \mathsf{JP}\}$
    $\mathsf{minJP} \longleftarrow \{t(\boldsymbol{u}, f, p) \mid t(\boldsymbol{u}, f, p) \in \mathsf{JP}, \deg(t\mathrm{lm}(f)) = d\}$
    $\mathsf{JP} \longleftarrow \mathsf{JP} \setminus \mathsf{minJP}$
    $\mathsf{minJP} \longleftarrow \mathsf{minJP} \setminus \{\text{J-pairs rejected by Syzygy or Rewriting Criterion}\}$
    $M \longleftarrow$ `SymbolicProcess`$(\mathsf{minJP}, G, \prec_p, \prec_s)$
    $M_I = \{t(\boldsymbol{u}, f, p) \in M \mid \mathrm{lm}(\boldsymbol{u}) \prec_s \boldsymbol{e}_l\}$
    $M_{II} = M \setminus M_I$
    # Eliminate Block I
    $\widetilde{M_I} \longleftarrow$ `OneDirectionElimination`$(\{t(\boldsymbol{u}, f) \mid t(\boldsymbol{u}, f, p) \in M_I\}, \prec_s)$
    # Eliminate Block II
    $M_{AB}, M_{CD} \longleftarrow$ `SubstituteBlock`$(\widetilde{M_I}, M_I, M_{II})$
    $\widetilde{M_{II}} \longleftarrow$ `FaugèreMethod`(M_{AB}, M_{CD})
    $G, \mathsf{JP}, H \longleftarrow$ `UpdateBlockI`$(\widetilde{M_I}, G, \mathsf{JP}, H, \prec_s)$
    $G, \mathsf{JP}, H \longleftarrow$ `UpdateBlockII`$(\widetilde{M_{II}}, G, \mathsf{JP}, H, \prec_s)$
  return $\{g \mid (\boldsymbol{v}, g, q) \in G\}$

Most of the 2-Block-GVW algorithm is similar to the Matrix-GVW except the one-direction elimination part. In 2-Block-GVW, the matrix M is divided into two blocks M_I and M_{II}. The block M_I is eliminated by general one-direction elimination at Step 14. We substitute polynomials in $\widetilde{M_I}$, put polynomials without new leading monomials to M_{AB} and those with new leading monomials into M_{CD}. Pairs in M_{II} are also appended to M_{CD}. This step is done by Function `SubstituteBlock(·)`. Faugère's method is used at Step 17, which is not detailed in current paper, and for more details please see [16]. Here, the result $\widetilde{M_{II}}$ refers to the polynomials (including 0) in the echelon form of the matrix $CA^{-1}B + D$ as well as their corresponding signatures, while the echelon form is obtained by using a dense one-direction elimination described in [22]. We update polynomials at Steps 18–19.

Function SubstituteBlock

Input : $\widetilde{M_I}$, a set of reduced pairs in $\boldsymbol{M}$; M_I, M_{II}, subsets of $\boldsymbol{M} \times R$.

Output: M_{AB}, M_{CD}, sets of pairs in $\boldsymbol{M}$.

```
1 begin
2     M_AB, M_new ⟵ ∅, ∅
3     for each (w, h) ∈ M̃_I with h ≠ 0 do
4         if ∃ t(u, f, p) ∈ M_I s.t. lm(tf) = lm(h) then
5             M_AB ⟵ M_AB ∪ {tp}
6         else
7             M_new ⟵ M_new ∪ {(w, h)}
8     M_CD ⟵ M_new ∪ {t(u, f) | t(u, f, p) ∈ M_II}
9     return M_AB, M_CD
```

In the function `SubstituteBlock()` pairs in $\widetilde{M_I}$ are divided into two kinds at Step 4. The first kind of pairs do not have new leading monomials, i.e., for $(\boldsymbol{w}, h) \in \widetilde{M_I}$, there exists $t(\boldsymbol{u}, f, p) \in M_I$ s.t. $\mathrm{lm}(tf) = \mathrm{lm}(h)$. In this case, the polynomial h is substituted by tp, which is considered to be sparser. The second kind of pairs having new leading monomials, are not substituted.

The function `UpdateBlockI(·)` updates the set G, `JP`, and H for Block I. A bit different from Function `Update(·)`, the above function needs to update 3-tuples in G. After the new pairs are collected in M_1^+ and M_2^+, we compute the reduced form of $\widetilde{M}$ by general Gaussian eliminations without consideration of signatures. We assume the polynomials in reduced echelon form is returned to the set E at Step 8. For sake of efficiency, we only compute the reduced forms for pairs in $M_1^+ \cup M_2^+$ in practical implementation at Step 8. Next, for each $(\boldsymbol{u}, f) \in M_1^+ \cup M_2^+$, we find out the polynomial $p \in E$ such that $\mathrm{lm}(f) = \mathrm{lm}(p)$, and regard $(\boldsymbol{u}, f, p)$ as a new 3-tuple. The other steps are the same as those in Function `Update(·)`.

Function UpdateBlockI

Input : $\widetilde{M}$, a set of reduced pairs in $\boldsymbol{M}$; G, a set of $\boldsymbol{M} \times R$; JP, a set of J-pairs; H, a set of syzygy pairs in $\boldsymbol{M}$; $\prec_s$, a monomial ordering on R^m.

Output: The updated sets of G, JP and H.

begin
  $M_1^+, M_2^+ \longleftarrow \emptyset, \emptyset$
  for every $(\boldsymbol{u}, f) \in \widetilde{M}$ with $f \neq 0$ **do**
    if $\nexists (\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ **then**
      $M_1^+ \longleftarrow M_1^+ \cup \{(\boldsymbol{u}, f)\}$
    else if $\frac{\mathrm{lm}(f)}{\mathrm{lm}(g)}\,\mathrm{lm}(\boldsymbol{v}) \succ_s \mathrm{lm}(\boldsymbol{u})$ *for* $\forall (\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g) | \mathrm{lm}(f)$ **then**
      $M_2^+ \longleftarrow M_2^+ \cup \{(\boldsymbol{u}, f)\}$
  $E \longleftarrow$ `ReducedEchelonForm`$(\widetilde{M})$ # E is a set of polynomials without signatures.
  for every $(\boldsymbol{u}, f) \in M_1^+ \cup M_2^+$ **do**
    Find p in E and $\mathrm{lm}(f) = \mathrm{lm}(p)$
    $\mathsf{JP} \longleftarrow \mathsf{JP} \cup \{\text{J-pairs of } (\boldsymbol{u}, f, p) \text{ and } (\boldsymbol{v}, g, q) \mid (\boldsymbol{v}, g, q) \in G\}$
    $G \longleftarrow G \cup \{(\boldsymbol{u}, f, p)\}$
  $H \longleftarrow H \cup \{(\boldsymbol{w}, 0) \mid (\boldsymbol{w}, 0) \in \widetilde{M}\}$
  return G, JP, H

Function UpdateBlockII

Input : $\widetilde{M}$, a set of reduced pairs in $\boldsymbol{M}$; G, a set of $\boldsymbol{M} \times R$; JP, a set of J-pairs; H, a set of syzygy pairs in $\boldsymbol{M}$; $\prec_s$, a monomial ordering on R^m.

Output: The updated sets of G, JP and H.

begin
  $M_1^+, M_2^+ \longleftarrow \emptyset, \emptyset$
  for every $(\boldsymbol{u}, f) \in \widetilde{M}$ with $f \neq 0$ **do**
    if $\nexists (\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g)$ divides $\mathrm{lm}(f)$ **then**
      $M_1^+ \longleftarrow M_1^+ \cup \{(\boldsymbol{u}, f)\}$
    else if $\frac{\mathrm{lm}(f)}{\mathrm{lm}(g)}\,\mathrm{lm}(\boldsymbol{v}) \succ_s \mathrm{lm}(\boldsymbol{u})$ *for* $\forall (\boldsymbol{v}, g) \in G$ s.t. $\mathrm{lm}(g) | \mathrm{lm}(f)$ **then**
      $M_2^+ \longleftarrow M_2^+ \cup \{(\boldsymbol{u}, f)\}$
  for every $(\boldsymbol{u}, f) \in M_1^+ \cup M_2^+$ **do**
    $\mathsf{JP} \longleftarrow \mathsf{JP} \cup \{\text{J-pairs of } (\boldsymbol{u}, f, f) \text{ and } (\boldsymbol{v}, g, q) \mid (\boldsymbol{v}, g, q) \in G\}$
    $G \longleftarrow G \cup \{(\boldsymbol{u}, f, f)\}$
  $H \longleftarrow H \cup \{(\boldsymbol{w}, 0) \mid (\boldsymbol{w}, 0) \in \widetilde{M}\}$
  return G, JP, H

The function `UpdateBlockII(·)` updates the set G, `JP`, and H for Block II. This function is more similar to Function `Update(·)`, and the only difference is it uses $(\boldsymbol{u}, f, f)$ instead of $(\boldsymbol{u}, f)$ in Steps 9–10.

Theorem 3.4 *The 2-Block-GVW algorithm computes a Gröbner basis for the input system.*

Proof For 2-Block-GVW algorithm, polynomials that are substituted all lie in Block I and they are used to reduce polynomials in Block II. So for each $(\boldsymbol{u}, f, p)$ of these polynomials, according to Steps 8–12 of Function `UpdateBlockI(·)`, we have that $f - p$ belongs to the linear span of $\{g \mid (\boldsymbol{v}, g, q) \in M_I\}$. So when tf is substituted by tp in later loops, we also have $t(f - p)$ belongs to the linear span of $\{g \mid (\boldsymbol{v}, g, q) \in M_I'\}$ where t is a monomial and M_I' is the corresponding matrix in the later loops. This is because all monomials that can be generated have been considered in Step 4 of Function `SymbolicProcess(·)`. ∎

For the k-Block-GVW algorithm Note that, we use 3-tuple $(\boldsymbol{u}, f, p)$ when the block number k is 2. In this case, we divide the ideal $I = \langle f_1, f_2, \cdots, f_m\rangle$ into $I_1 = \langle f_1, \cdots, f_{l-1}\rangle$ and $I_2 = \langle f_l, \cdots, f_m\rangle$. We always have $f - p \in I_1$ for all pairs. So we substitute $f \longrightarrow p$ when eliminating Block II, and this does not affect the correctness of the algorithm.

Next, we consider more blocks, i.e., $k > 2$, and we need to use $k+1$-tuple $(\boldsymbol{u}, f, p_1, \cdots, p_{k-1})$. For example, let $k = 4$, then the tuple is $(\boldsymbol{u}, f, p_1, p_2, p_3)$. Assume the ideal I is divided into $I_1 = \langle f_1, \cdots, f_{l-1}\rangle$, $I_2 = \langle f_l, \cdots, f_{s-1}\rangle$, $I_3 = \langle f_s, \cdots, f_{t-1}\rangle$, and $I_4 = \langle f_t, \cdots, f_m\rangle$. Now we will always have $f - p_1 \in I_1$, $f - p_2 \in I_1 \cup I_2$, and $f - p_3 \in I_1 \cup I_2 \cup I_3$. When eliminating Block $i+1$, we substitute $f \longrightarrow p_i$. In this case, the update of p_i becomes complex, but still in a similar way to the 2-Block-GVW algorithm.

3.4 The LM-GVW Algorithm

We present the LM-GVW algorithm in this subsection. The basic idea is similar to Block-GVW, but we substitute all polynomials in LM-GVW not only polynomials in some blocks. The goal of LM-GVW algorithm is trying to find the mutants before one-direction eliminating the whole matrix. The main idea is illustrated in Figure 4 of Section 2.

First, we give the definition of mutants.

Definition 3.5 Let $(\boldsymbol{u}, f)$ be a pair in $\boldsymbol{M}$ which is generated by $\{(\boldsymbol{e}_i, f_i) \mid 1 \le i \le m\}$. Then we say $(\boldsymbol{u}, f)$ is a mutant if $\deg(f) < \max\{\deg(p_i f_i) \mid 1 \le i \le m\}$ where $\boldsymbol{u} = (p_1, \cdots, p_m)$.

Clearly, if f is a linear polynomial, $(\boldsymbol{u}, f)$ is a mutant.

In LM-GVW algorithm, the 3-tuple $(\boldsymbol{u}, f, q)$ is also used instead of $(\boldsymbol{u}, f)$. To make a difference from the notation used in the last subsection, we use q instead of p. The polynomial q is used to substitute f, and it is an only auxiliary polynomial. Initially, we set $q := f$, and update q in Function `UpdateLM(·)`. Concepts defined for $(\boldsymbol{u}, f)$ can also be extended to $(\boldsymbol{u}, f, q)$ by ignoring q.

The LM-GVW algorithm is described below.

Algorithm 4: The LM-GVW algorithm

Input : $f_1, \cdots, f_m \in R$; $\prec_p$ and $\prec_s$, monomial orderings for R and R^m.

Output: A Gröbner basis of $I = \langle f_1, \cdots, f_m \rangle$.

1 **begin**
2 $G \longleftarrow \{(\boldsymbol{e}_i, f_i, f_i) \mid 1 \le i \le m\}$
3 $H \longleftarrow \{(f_i \boldsymbol{e}_i - f_i \boldsymbol{e}_j, 0) \mid 1 \le i < j \le m\}$
4 $\mathsf{JP} \longleftarrow$ all J-pairs of G
5 **while** $\mathsf{JP} \neq \emptyset$ **do**
6 $d \longleftarrow \min\{\deg(t\mathrm{lm}(f)) \mid (t(\boldsymbol{u}, f, q) \in \mathsf{JP}\}$
7 $\mathsf{minJP} \longleftarrow \{t(\boldsymbol{u}, f, q) \mid t(\boldsymbol{u}, f, q) \in \mathsf{JP}, \deg(t\mathrm{lm}(f)) = d\}$
8 $\mathsf{JP} \longleftarrow \mathsf{JP} \setminus \mathsf{minJP}$
9 $\mathsf{minJP} \longleftarrow \mathsf{minJP} \setminus \{\text{J-pairs rejected by Syzygy or Rewriting Criterion}\}$
10 $M \longleftarrow \texttt{SymbolicProcess}(\mathsf{minJP}, G, \prec_p, \prec_s)$
11 # Elimination for new leading monomials
12 $M_{AB} \longleftarrow \{tq \mid t(\boldsymbol{u}, f, q) \in M \setminus \mathsf{minJP}\}$
13 $M_{CD} \longleftarrow \{tf \mid t(\boldsymbol{u}, f, q) \in \mathsf{minJP}\}$
14 $\widetilde{L} \longleftarrow \texttt{Faugère Method}(M_{AB}, M_{CD})$
15 **if** a set P of mutants are discovered in $\widetilde{L}$ **then**
16 $F \longleftarrow$ inner-reduce $\{f_1, \cdots, f_m\} \cup P$
17 **return** $\texttt{LM-GVW}(F, \prec_p, \prec_s)$
18 # One direction elimination
19 $\widetilde{M} \longleftarrow \texttt{OneDirectionElimination}(\{t(\boldsymbol{u}, f) \mid t(\boldsymbol{u}, f, q) \in M\}, \prec_s)$
20 $G, \mathsf{JP}, H \longleftarrow \texttt{UpdateLM}(\widetilde{M}, G, \mathsf{JP}, H, \prec_s)$
21 **return** $\{g \mid (\boldsymbol{v}, g, p) \in G\}$

We skip the steps that are similar to Matrix-GVW and focus on Steps 11–20. After symbolic process, we substitute polynomials in M. Specifically, we substitute $f \longrightarrow q$ for polynomials in $M \backslash \texttt{minJP}$ and stored in M_{AB}. Polynomials in `minJP` are not substituted and kept in M_{CD} without signatures. Next, Faugère's method is used at Step 14. Again, the result $\widetilde{L}$ refers to the set of polynomials in the echelon form of the matrix $CA^{-1}B + D$. If mutants appear, we recursive call LM-GVW. Otherwise, we do general one-direction elimination, and update the intermediate sets at Step 20. Here, we reuse the function `UpdateBlockI(·)` as `UpdateLM(·)`.

The LM-GVW algorithm tries to find mutants, including linear polynomials, before one-direction eliminating the whole matrix. Clearly, all polynomials in $\widetilde{L}$ belong to the ideal $I = \langle f_1, \cdots, f_m \rangle$. Besides, the polynomial q in 3-tuple $(\boldsymbol{u}, f, q)$ never affect the one-direction elimination of $(\boldsymbol{u}, f)$. So the correctness of LM-GVW is straightforward.

According to our experimental results, we have the following conjuncture, but we are unable to prove it in theory.

Conjuncture *The linear span* $\langle M_{AB} \cup M_{CD} \rangle_k$ *is exactly the same as the linear span* $\langle \{tf \mid t(\boldsymbol{u}, f, q) \in M\} \rangle_k = \langle \{h \mid (\boldsymbol{w}, h) \in \widetilde{M}\} \rangle_k$ *in each loop, where* $\langle F \rangle_k$ *is the linear space*

spanned by F over the field k.

3.5 The Sub-GVW Algorithm

Combining the Block-GVW algorithm and the LM-GVW algorithm, we get the Sub-GVW algorithm. Specifically, in each main loop, the Sub-GVW algorithm first performs Steps 11–17 in the LM-GVW algorithm. If mutants, say P, are discovered from the eliminating results, then the Sub-GVW algorithm is recursive called for the set $\{f_1, \cdots, f_m\} \cup P$ after inter-reduction. Otherwise, when no mutants are found, the Sub-GVW algorithm performs one direction eliminations by dividing the matrix into blocks, e.g., Steps 11–17 in the 2-Block-GVW algorithm are done. After the set G, JP, and auxiliary polynomials are updated respectively, the Sub-GVW algorithm starts the next main loop.

3.6 A Toy Example

To illustrate the Sub-GVW algorithms, we give a toy example. We consider the system used by Faugère[6].

Example 3.6 Let $\{f_1, f_2, f_3\} \cup R = \mathbb{F}_{23}[x_1, x_2, x_3]$ be a set of polynomials, and $f_1 = x_1^2 + 18x_1x_2 + 19x_2^2 + 8x_1x_3 + 5x_2x_3 + 7x_3^2$, $f_2 = 3x_1^2 + 7x_1x_2 + 8x_2^2 + 22x_1x_3 + 11x_2x_3 + 22x_3^2$, $f_3 = 6x_1^2 + 12x_1x_2 + 4x_2^2 + 14x_1x_3 + 9x_2x_3 + 7x_3^2$. Monomial ordering $\prec_p$ is the Graded Reverse Lexicographic ordering with $x_3 < x_2 < x_1$, and $\prec_s$ in R^3 is a position over term extension of $\prec_p$, i.e.

$$x^\alpha \boldsymbol{e}_i \prec_s x^\beta \boldsymbol{e}_j \text{ iff } i < j, \text{ or } i = j \text{ and } x^\alpha \prec_p x^\beta.$$

Thus, we have $\boldsymbol{e}_1 \prec_s \boldsymbol{e}_2 \prec_s \boldsymbol{e}_3$.

We compute the Gröbner basis of $\langle f_1, f_2, f_3\rangle$ using the Sub-GVW algorithm. We only construct the matrice of degrees 2 and 3 to show our algorithm. The complete procedure is complicated and omitted here. For simplicity, we usually do not show all four sub-matrices used in Faugère's method.

Initially, we have $G = \{(\boldsymbol{e}_1, f_1), (\boldsymbol{e}_2, f_2), (\boldsymbol{e}_3, f_3)\}$ and $JP = \{(\boldsymbol{e}_2, f_2), (\boldsymbol{e}_3, f_3)\}$. We divide the matrices in each loop into *two* blocks and use 2-Block-GVW. That is, let $\boldsymbol{M}$ be the module generated by G. Then denote M_I be the module generated by $\{(\boldsymbol{e}_1, f_1), (\boldsymbol{e}_2, f_2)\}$, and $M_{II} := M \setminus M_I$.

Since we divide the matrices into two blocks, we need an auxiliary polynomial, say p, to do substitutions in Block I. We also need an auxiliary polynomial q to substitute polynomials in the LM-GVW algorithm. So for each pair $(\boldsymbol{u}, f)$ in this Sub-GVW, it has two auxiliary polynomials. At the beginning, we have $f_i = p_i = q_i$ for $i = 1, 2, 3$.

Degree 2 As all the auxiliary polynomials are the same as f_i, there are no substitutions in both Block-GVW and LM-GVW. The matrix of degree 2 is

$$M^{(2)} = \begin{array}{c} \\ (\boldsymbol{e}_1, f_1) \\ (\boldsymbol{e}_2, f_2) \\ (\boldsymbol{e}_3, f_3) \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & 18 & 19 & 8 & 5 & 7 \\ 3 & 7 & 8 & 22 & 11 & 22 \\ 6 & 12 & 4 & 14 & 9 & 7 \end{pmatrix} \end{array}.$$

LM-GVW. By doing Steps 11–17 in LM-GVW, no mutants are found. Next, we need to do one-direction elimination to $M^{(2)}$ using Steps 11–17 in 2-Block-GVW.

Eliminating Block I. In this example, the matrix $M_I^{(2)}$ is constructed by the first two rows in the $M^{(2)}$. After doing one-direction elimination, we get the matrix $\widetilde{M}_I^{(2)}$:

$$\widetilde{M}_I^{(2)} = \begin{array}{c} \\ (\boldsymbol{e}_1, f_1) \\ (\boldsymbol{e}_2, f_4) \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & 18 & 19 & 8 & 5 & 7 \\ 0 & 1 & 3 & 2 & 4 & 22 \end{pmatrix} \end{array}.$$

From the echelon form, we find a polynomial with new leading monomial denoted as f_4, whose signature is $\boldsymbol{e}_2$.

Eliminating Block II. To obtain the eliminating result $\widetilde{M}_{II}^{(2)}$, we substitute the polynomials in $\widetilde{M}_I^{(2)}$. The polynomial f_1 does not have new leading monomial so it is substituted and put into $M_{AB}^{(2)}$:

$$M_{AB}^{(2)} = f_1 \begin{array}{c} \begin{array}{ccccccc} x_1^2 & & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & | & 18 & 19 & 8 & 5 & 7 \end{pmatrix} \end{array}.$$

The polynomial f_4 has a new leading monomial. Together with $(\boldsymbol{e}_3, f_3)$, we get the matrix $M_{CD}^{(2)}$:

$$M_{CD}^{(2)} = \begin{array}{c} \\ (\boldsymbol{e}_2, f_4) \\ (\boldsymbol{e}_3, f_3) \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1^2 & & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 0 & | & 1 & 3 & 2 & 4 & 22 \\ 6 & | & 12 & 4 & 14 & 9 & 7 \end{pmatrix} \end{array}.$$

We eliminate the $M_{II}^{(2)}$ using Faugère method. The matrix of $\widetilde{M}_{II}^{(2)}$ is

$$\widetilde{M}_{II}^{(2)} = \begin{array}{c} \\ (\boldsymbol{e}_3, f_5) \end{array} \begin{array}{c} \begin{array}{ccccc} x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & 3 & 2 & 4 & 22 \\ 0 & 1 & 12 & 20 & 18 \end{pmatrix} \end{array}.$$

In matrix $\widetilde{M}_{II}^{(2)}$, the polynomial f_5 has the new leading monomial and its signature is $\boldsymbol{e}_3$.

Updating. So far, the one-direction elimination of $M^{(2)}$ has been done, and we need to do some updates for further computations.

We compute the reduced form of $\widetilde{M}_I^{(2)}$ and obtain the matrix $E^{(2)}$.

$$E^{(2)} = \begin{array}{c} \\ p_1' \\ p_4 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & 0 & 11 & 18 & 2 & 2 \\ 0 & 1 & 3 & 2 & 4 & 22 \end{pmatrix} \end{array}.$$

We compute the reduced form for the whole matrix, and get

$$\begin{array}{c} \\ q_1' \\ q_4 \\ q_5 \end{array} \begin{array}{c} \begin{array}{cccccc} x_1^2 & x_1x_2 & x_2^2 & x_1x_3 & x_2x_3 & x_3^2 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 12 & 11 \\ 0 & 1 & 0 & 12 & 13 & 14 \\ 0 & 0 & 1 & 12 & 20 & 18 \end{pmatrix} \end{array}.$$

Now the set G is updated to $\{(\boldsymbol{e}_1, f_1, p'_1, q'_1), (\boldsymbol{e}_2, f_2, p_2, q_2), (\boldsymbol{e}_3, f_3, p_3, q_3), (\boldsymbol{e}_2, f_4, p_4, q_4), (\boldsymbol{e}_3, f_5, p_5 = f_5, q_5)\}$. Note that the auxiliary polynomials of f_2 and f_3 are not updated.

Degree 3 The matrix of degree 3 is

$$M^{(3)} = \begin{array}{c} \\ x_2(\boldsymbol{e}_1, f_1) \\ x_3(\boldsymbol{e}_1, f_1) \\ x_1(\boldsymbol{e}_2, f_4) \\ x_2(\boldsymbol{e}_2, f_4) \\ x_3(\boldsymbol{e}_2, f_4) \\ x_1(\boldsymbol{e}_3, f_5) \\ x_2(\boldsymbol{e}_3, f_5) \\ x_3(\boldsymbol{e}_3, f_5) \end{array} \begin{array}{c} \begin{array}{ccccccccc} x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 18 & 19 & 0 & 8 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 1 & 18 & 19 & 8 & 5 & 7 \\ 1 & 3 & 0 & 2 & 4 & 0 & 22 & 0 & 0 \\ 0 & 1 & 3 & 0 & 2 & 4 & 0 & 22 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 2 & 4 & 22 \\ 0 & 1 & 0 & 12 & 20 & 0 & 18 & 0 & 0 \\ 0 & 0 & 1 & 0 & 12 & 20 & 0 & 18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 12 & 20 & 18 \end{pmatrix} \end{array}.$$

LM-GVW. We construct the matrix of degree 3 in LM-GVW, where all the polynomials are substituted with corresponding auxiliary polynomial except for those in J-pairs ($x_1(\boldsymbol{e}_2, f_4)$ and $x_1(\boldsymbol{e}_3, f_5)$). Thus, we have the matrix:

$$\begin{array}{c} \\ x_2q'_1 \\ x_3q'_1 \\ x_1(\boldsymbol{e}_2, f_4) \\ x_2q_4 \\ x_3q_4 \\ x_1(\boldsymbol{e}_3, f_5) \\ x_2q_5 \\ x_3q_5 \end{array} \begin{array}{c} \begin{array}{ccccccccc} x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 12 & 0 & 11 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 12 & 11 \\ 1 & 3 & 0 & 2 & 4 & 0 & 22 & 0 & 0 \\ 0 & 1 & 0 & 0 & 12 & 13 & 0 & 14 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 12 & 13 & 14 \\ 0 & 1 & 0 & 12 & 20 & 0 & 18 & 0 & 0 \\ 0 & 0 & 1 & 0 & 12 & 20 & 0 & 18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 12 & 20 & 18 \end{pmatrix} \end{array}.$$

Note that we do not know the exact signatures of the substituted polynomials, so we omit them in the above matrix. The above matrix is sparser than $M^{(3)}$, and leads to a more efficient elimination. The result of Faugère's method is as follows:

$$\widetilde{L}^{(3)} = \begin{array}{c} \begin{array}{ccccccccc} x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 22 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 22 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 18 \end{pmatrix} \end{array}.$$

As no mutant polynomials appear, the block-GVW is executed.

Eliminating Block I. We construct the matrix $M_I^{(3)}$:

$$M_I^{(3)} = \begin{array}{c} \\ x_2(\boldsymbol{e}_1, f_1) \\ x_3(\boldsymbol{e}_1, f_1) \\ x_1(\boldsymbol{e}_2, f_4) \\ x_2(\boldsymbol{e}_2, f_4) \\ x_3(\boldsymbol{e}_2, f_4) \end{array} \begin{array}{c} \begin{array}{ccccccccc} x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 18 & 19 & 0 & 8 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 1 & 18 & 19 & 8 & 5 & 7 \\ 1 & 3 & 0 & 2 & 4 & 0 & 22 & 0 & 0 \\ 0 & 1 & 3 & 0 & 2 & 4 & 0 & 22 & 0 \\ 0 & 0 & 0 & 0 & 1 & 3 & 2 & 4 & 22 \end{pmatrix} \end{array}.$$

After the one-direction elimination, we obtain the matrix $\widetilde{M}_I^{(3)}$:

$$\widetilde{M}_I^{(3)} = \begin{array}{c} \\ x_2(\boldsymbol{e}_1, f_1) \\ (x_1\boldsymbol{e}_2, f_6) \\ (x_2\boldsymbol{e}_2, f_7) \\ x_3(\boldsymbol{e}_1, f_1) \\ x_3(\boldsymbol{e}_2, f_4) \end{array} \begin{array}{c} \begin{array}{ccccccccc} x_1^2x_2 & x_1x_2^2 & x_2^3 & x_1^2x_3 & x_1x_2x_3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 18 & 19 & 0 & 8 & 5 & 0 & 7 & 0 \\ 0 & 8 & 4 & 2 & 19 & 18 & 22 & 16 & 0 \\ 0 & 0 & 14 & 17 & 14 & 19 & 3 & 20 & 0 \\ 0 & 0 & 0 & 1 & 18 & 19 & 8 & 5 & 7 \\ 0 & 0 & 0 & 0 & 1 & 3 & 2 & 4 & 22 \end{pmatrix} \end{array},$$

where f_6 and f_7 are new polynomials.

Eliminating Block II. The polynomials without new leading monomials are at row 1, 2, 3, 5, and these polynomials are substituted by auxiliary polynomials. The polynomial f_7 has a new leading monomial is unsubstituted, and put into $M_{CD}^{(3)}$. Then to eliminate the matrix $M_{II}^{(3)}$, we get $M_{AB}^{(3)}$:

$$M_{AB}^{(3)} = \begin{array}{c} \\ x_2p_1 \\ x_2p_4 \\ x_3p_1 \\ x_3p_4 \end{array} \begin{array}{c} \begin{array}{cccc|ccccc} x_1^2x_2 & x_1x_2^2 & x_1^2x_3 & x_1x_2x_3 & x_2^3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \left(\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0 & 0 & 21 & 22 & 8 & 14 \\ 0 & 1 & 0 & 2 & 3 & 4 & 0 & 22 & 0 \\ 0 & 0 & 1 & 0 & 0 & 21 & 22 & 8 & 14 \\ 0 & 0 & 0 & 1 & 0 & 3 & 2 & 4 & 22 \end{array}\right) \end{array}$$

and the matrix $M_{CD}^{(3)}$:

$$M_{CD}^{(3)} = \begin{array}{c} \\ (x_2\boldsymbol{e}_2, f_7) \\ x_1(\boldsymbol{e}_3, f_5) \\ x_2(\boldsymbol{e}_3, f_5) \\ x_3(\boldsymbol{e}_3, f_5) \end{array} \begin{array}{c} \begin{array}{cccc|ccccc} x_1^2x_2 & x_1x_2^2 & x_1^2x_3 & x_1x_2x_3 & x_2^3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \left(\begin{array}{cccc|ccccc} 0 & 0 & 17 & 14 & 14 & 19 & 3 & 20 & 0 \\ 0 & 1 & 12 & 20 & 0 & 0 & 18 & 0 & 0 \\ 0 & 0 & 0 & 12 & 1 & 20 & 0 & 18 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 12 & 20 & 18 \end{array}\right) \end{array}.$$

Please note that the matrix $M_{AB}^{(3)}$ is reduced to echelon form and it is much sparser than $\widetilde{M}_I^{(3)}$.

At last, we eliminate the $M_{CD}^{(3)}$ via Faugère's method and get the matrix

$$\widetilde{M}_{II}^{(3)} = \begin{array}{c} \\ (x_1\boldsymbol{e}_3, f_8) \\ (x_2\boldsymbol{e}_3, f_9) \\ (x_3\boldsymbol{e}_3, f_{10}) \end{array} \begin{array}{c} \begin{array}{ccccc} x_2^3 & x_2^2x_3 & x_1x_3^2 & x_2x_3^2 & x_3^3 \end{array} \\ \begin{pmatrix} 1 & 8 & 1 & 11 & 15 \\ 0 & 1 & 14 & 22 & 6 \\ 0 & 0 & 1 & 17 & 6 \\ 0 & 0 & 0 & 1 & 18 \end{pmatrix} \end{array}.$$

Now we have 3 more new polynomials: f_8, f_9, f_{10}.

Updating. Totally, five new polynomials, $f_{6,\cdots,10}$, are generated. We can compute p_6 and p_7 through computing the reduced echelon form of $\widetilde{M}_I^{(3)}$, and $p_i = f_i$ for $i = 8, 9, 10$. For q_i's where $i = 6, \cdots, 10$, we can find them from the matrix $\widetilde{L}^{(3)}$, because Conjuncture in Subsection 3.4 implies $\widetilde{L}^{(3)}$ always be a linear basis of the linear span generated by rows $M^{(3)}$.

4 Experimental Results

We implemented the Sub-GVW alogrithm over Boolean polynomial rings in C++. The complied codes will be available online sooner.

To show the performance of our substituting methods, we compare the densities of the matrices before and after substitutions, and we also present the eliminating time for those matrices. For the systems to be tested with, we use the "MQn" System[25], where each MQn System consists of n quadratic polynomials in n variables. The MQ systems are more similar to randomly generated systems as well as most cipher systems, so they will reflect more generic features. We also consider detailed matrices with different degrees in each MQn System. For instance, "MQ24_6" means the degree-6 matrix generated during the computation of the MQ24 System. In these experiments, the 2-Block-GVW algorithm is used, i.e., each matrix in the main loop is divided into 2 blocks. The experimental platform is MacBook Pro with 2.6 GHz Intel Core i7, 16 GB memory.

In Table 1, we compare the densities of Block I in three cases: Before one-direction elimination, after one-direction elimination, and after one-direction elimination and substitutions.

From Table 1, we can see that the matrices become much sparser after substitutions. The matrices of Block I become even sparser than the matrices before the one-direction elimination. The differences of density directly results in the differences of eliminating time of Block II. In Table 2, we present the time of eliminating Block II for two cases: Using polynomials in the echelon form of the one-direction elimination of Block I, and using polynomials after substitutions. Faugère's method is used for eliminating Block II. The time in the table is given in seconds. The last column is the times of speeding up.

Table 1 Density of Block I

Matrix	Density of Block I		
	before reduction	after reduction	after substitution
MQ24_5	2.70%	10.80%	2.30%
MQ25_5	2.65%	10.55%	2.22%
MQ26_5	2.46%	9.70%	2.07%
MQ27_5	2.32%	9.37%	1.97%
MQ24_6	2.03%	8.96%	1.53%
MQ25_6	2.04%	8.77%	1.50%
MQ26_6	2.01%	8.41%	1.47%
MQ27_6	1.88%	8.27%	1.41%

Table 2 Time for eliminating Block II

Matrix	Time for eliminating block II		speedup
	no substitution	with substitution	
MQ24_5	11.67 s	4.94 s	2.36
MQ25_5	14.94 s	7.45 s	2.00
MQ26_5	20.43 s	12.75 s	1.60
MQ27_5	31.81 s	19.51 s	1.63
MQ24_6	98.35 s	51.76 s	1.90
MQ25_6	249.84 s	108.43 s	2.30
MQ26_6	544.92 s	248.92 s	2.19
MQ27_6	1193.07 s	477.73 s	2.50

Table 2 shows that the lower density does speed up the eliminating time of Block II.

In Table 3, we compare the density of the whole matrices before and after substitutions in the LM-GVW algorithm. The matrices in consider are all before eliminations. Again, the matrices become much sparser after substitutions. The eliminating timings are given in Table 4.

In Table 4, we list the time for three methods of eliminating matrices: Pure dense one-direction elimination[22], one-direction elimination by using blocks and substituting techniques, and elimination done in LM-GVW after substitutions. These three methods are denoted as Dense method, Block method, and LM method in Table 4 respectively. The timings are given in seconds.

Table 3 Density of matrix

Matrix	before substitution	after substitution
MQ24_5	3.90%	2.96%
MQ25_5	3.73%	2.87%
MQ26_5	3.53%	2.74%
MQ27_5	3.33%	2.66%
MQ24_6	3.35%	1.20%
MQ25_6	3.29%	1.33%
MQ26_6	3.25%	1.41%
MQ27_6	3.08%	1.50%

Table 4 Times for reducing matrix

Matrix	Dense method	Block method	LM method
MQ24_5	41.79 s	16.14 s	1.94 s
MQ25_5	71.29 s	26.18 s	3.34 s
MQ26_5	112.71 s	38.57 s	5.65 s
MQ27_5	181.91 s	55.85 s	9.58 s
MQ24_6	1633.33 s	317.89 s	17.27 s
MQ25_6	3595.68 s	639.33 s	43.57 s
MQ26_6	>1 h	1295.80 s	115.71 s
MQ27_6	>1 h	2044.39 s	299.83 s

Results in Table 4 show that the block method is much more efficient than the dense one-direction method, because sparse linear algebra and Faugère's method are used. Besides, we can find that eliminations done in LM-GVW are much efficient than those done by the Block method. This is because LM-GVW eliminates the substituted matrices in a two-direction way. So LM-GVW is efficient for finding linear or mutants polynomials before the one-direction elimination is done.

At last, we compare our implementation of Sub-GVW with some public softwares. We consider the Gröbner basis routines on Magma (ver. 2.20-3), PolyBori (ver. 6.4.1), and Maple (ver. 18). Specially, there have been many improvements for Boolean Rings since Magma ver. 2.19. Besides the MQ systems, we also consider some systems from cryptanalysis problems.

- Present_5r: A polynomial system originated from the key recovery problem of the block cipher Present with one pair of known plaintext and ciphertext[26]. Here a reduced version of Present which has only 5 rounds is considered. By setting the 80-bit key as variables and adding some internal variables used to simplify the structure of the systems, we generate a system with degree 2. Then we randomly guess the value of the first 48 variables, and

after substitution, we obtain the input polynomial system of our experiments. This is a quadratic system with 123 variables and 875 polynomials.

- Maya: A polynomial system originated from the problem of recovering a secret 4-bit Sbox, $S : GF(2)^4 \to GF(2)^4$, of the block cipher Maya from its input and output differences[27, 28]. The variables of this system are corresponding to the different bits of 16 bytes output $S(0000), S(0001), \cdots, S(1111)$, hence the system has $16 \times 4 = 64$ variables. In this system, there are 64 quadratic polynomials which represent the input and output differences of the Sbox, and other polynomials which represent the bijection property of the Sbox have degree not bigger than 4.

- Trivium: A polynomial system originated from the problem of recovering the internal states of the stream cipher Trivium[29, 30]. By setting the 288-bit internal states as variables, and guess the value of 113 variables, we can generate the input system after substitution. This system has 31 variables and 56 polynomials with degree 2.

- Bivium: A polynomial system originated from the problem of recovering the internal states of the stream cipher Bivium[30, 31], which is a reduced version of the stream cipher Trivium. We set the 177-bit internal states of Bivium as variables, and guess the value of 32 variables. After substitution, we generate the input system which has 33 variables and 48 polynomials with degree 2.

- Serpent: A polynomial system originated from the problem of recovering the initial key from the 2-round key schedule of the block cipher Serpent. This problem is the basic solving problem of the Cold Boot key recovery problem of Serpent[32]. The input system has 128 variables which corresponding to the 128-bit initial key, and 256 polynomials with degree 3.

- Canfil10: A polynomial system originated from the problem of recovering the internal states of a stream cipher based on a linear feedback shift register (LFSR) and a filter function[33, 34], and has the form

$$\{f(x_1, x_2, \cdots, x_n), f(L(x_1, x_2, \cdots, x_n)), \cdots, f(L^{m-1}(x_1, x_2, \cdots, x_n))\}.$$

  For this problem, $n = 100$, $m = 141$, $L(x_1, x_2, \cdots, x_{100}) = (x_2, x_3, \cdots, x_{100}, x_{38} + x_1)$, $f(x_1, x_2, \cdots, x_{100}) = x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_5 + x_6x_7 + x_1 + x_2 + x_3$.

The Sub-GVW algorithm in these experiments is implemented by integrating 3-Block-GVW and LM-GVW. The experimental platform is MacBook Pro with 2.6 GHz Intel Core i7, 16 GB memory. The timings are given in Table 5 in seconds.

Table 5 Comparisons with public softwares

System	Magma (ver. 2.20-3)	PolyBori (ver. 6.4.1)	Maple (ver18 FGb)	Sub-GVW
MQ24	100.60 s	610.74s	>1 h	46.68 s
MQ26	306.57 s	–	>1 h	156.01 s
MQ28	1169.15 s	–	>1 h	404.97 s
Present_5r	527.72 s	–	>1 h	82.80 s
Maya	19.36 s	–	162.39 s	12.54 s
Trivium	141.87 s	–	1971.29 s	7.23 s
Bivium	63.73 s	>1 h	3569.79 s	14.55 s
Serpent	168.66 s	192.25 s	108.85 s	4.05 s
Canfil10	11.69 s	23.37 s	3250.29 s	11.45 s

Results from the above table show that our implementation of the Sub-GVW algorithm is comparable with most of public Gröbner basis routines.

5 Conclusions

In this paper, we improve the GVW algorithm by using a substituting method. Two substituting ways are presented and integrated in the Sub-GVW algorithm. Different from the original GVW, the improved algorithm stores more auxiliary polynomials. These auxiliary polynomials are used for substitutions, which make the matrices sparser hence accelerate the speed of eliminating. Experimental results show this substituting method does improve the efficiency of eliminating matrices. Besides, our implementation is also shown comparable to most of public Gröbner basis routines.

We find the conjuncture in Subsection 3.4 always holds in our experiments. But we are not able to prove the correctness of this conjuncture at present. We will try to prove it in our future work.

References

[1] Buchberger B, Ein Algorithmus zum auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal), PhD thesis, University of Innsbruck, Innsbruck, Austria, 1965; English translation in *Journal of Symbolic Computation*, 2006, **41**(3–4): 475–511.

[2] Lazard D, Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations, *Proc. EUROCAL'83, Lect. Notes in Comp. Sci.*, 1983, **162**: 146–156.

[3] Faugère J C, A new effcient algorithm for computing Gröbner bases (F_4), *J. Pure Appl. Algebra*, 1999, **139**(1–3): 61–88.

[4] Courtois N, Klimov A, Patarin J, et al., Efficient algorithms for solving overdefined systems of multivariate polynomial equations, *Proc. of EUROCRYPT'00, Lect. Notes in Comp. Sci.,* 2000, **1807**: 392–407.

[5] Ding J, Buchmann J, Mohamed M S E, et al., MutantXL, *Proc. SCC'08*, 2008, 16–22.

[6] Faugère J C, A new effcient algorithm for computing Gröbner bases without reduction to zero (F_5), *Proc. ISSAC'02*, ACM Press, 2002, 75–82, Revised version downloaded from fgbrs.lip6.fr/jcf/Publications/index.html.

[7] Eder C and Perry J, F5C: A variant of Faugère's F5 algorithm with reduced Gröbner bases, *J. Symb. Comput.,* 2010, **45**(12): 1442–1458.

[8] Hashemi A and Ars G, Extended F5 criteria, *J. Symb. Comput.,* 2010, **45**(12): 1330–1340.

[9] Arri A and Perry J, The F5 criterion revised, *J. Symb. Comput.*, 2011, **46**: 1017–1029.

[10] Eder C and Roune B H, Signature rewriting in Gröbner basis computation, *Proc. ISSAC'13*, ACM Press, New York, USA, 2013, 331–338.

[11] Gao S H, Guan Y H, and Volny F, A new incremental algorithm for computing Gröbner bases, *Proc. ISSAC'10*, ACM Press, New York, USA, 2010, 13–19.

[12] Gao S H, Volny F, and Wang M S, A new framework for computing Gröbner bases, *Mathematics of Computation,* 2016, **85**(297): 449–465.

[13] Sun Y and Wang D K, A generalized criterion for signature related Gröbner basis algorithms, *Proc. ISSAC'11*, ACM Press, 2011, 337–344.

[14] Sun Y, Wang D K, Ma D X, et al., A signature-based algorithm for computing Gröbner bases in solvable polynomial algebras, *Proc. ISSAC'12*, ACM Press, 2012, 351–358.

[15] Boyer B, Eder C, Faugère J, et al., GBLA: Gröbner basis linear algebra package, *ACM on International Symposium on Symbolic and Algebraic Computation*, 2016.

[16] Faugère J C and Lachartre S, Parallel Gaussian elimination for Gröbner bases computations in finite fields, *Proc. PASCO*, ACM Press, 2010, 89–97.

[17] Albrecht M and Perry J, F4/5, Preprint, arXiv: 1006.4933v2 [math.AC], 2010.

[18] Bardet M, Faugère J C, and Salvy B, On the complexity of the F5 Gröbner basis algorithm, arXiv: 1312.1655, 2013.

[19] Faugère J C and Rahmany S, Solving systems of polynomial equations with symmetries using SAGBI-Gröbner bases, *Proc. ISSAC'09*, ACM Press, New York, USA, 2009, 151–158.

[20] Roune B H and Stillman M, Practical Gröbner basis computation, *Proc. ISSAC'12*, ACM Press, 2012.

[21] Boyer B, Eder C, Faugère J C, et al., GBLA: Gröbner basis linear algebra package, *International Symposium on Symbolic and Algebraic Computation*, 2016, 135–142.

[22] Sun Y, Lin D D, and Wang D K, An improvement over the GVW algorithm for inhomogeneous polynomial systems, *Finite Fields and Their Applications,* 2016, **41**: 174–192.

[23] Albrecht M and Bard G, The M4RI Library — Version 20130416, 2013, http://m4ri.sagemath.org.

[24] Sun Y, Lin D D, and Wang D K, On implementing the symbolic preprocessing function over Boolean polynomial rings in Gröbner basis algorithms using linear algebra, *Journal of Systems Science and Complexity,* 2016, **29**(3): 789–804.

[25] Courtois N, Benchmarking algebraic, logical and constraint solvers and study of selected hard problems, 2013, http://www.cryptosystem.net/aes/hardproblems.html.

[26] Bogdanov A, Knudsen L R, Leander G, et al., Present: An ultra-lightweight block cipher, *Cryptographic Hardware and Embedded Systems — CHES*, Springer, Berlin Heidelberg, 2007, 450–466.

[27] Borghoff J, Knudsen L R, Leander G, et al., Slender-set differential cryptanalysis, *Journal of Cryptology*, 2013, **26**(1): 11–38.

[28] Liu G Q and Jin C H, Differential cryptanalysis of PRESENT-like cipher, *Designs, Codes and Cryptography,* 2015, **76**(3): 385–408.

[29] Cannière C De, Trivium: A stream cipher construction inspired by block cipher design principles, *International Conference on Information Security,* Springer Berlin Heidelberg, 2006, 171–186.

[30] Huang Z and Lin D, Attacking bivium and trivium with the characteristic set method, *Progress in Cryptology — AFRICACRYPT* 2011, LNCS, 2011, **6737**: 77–91.

[31] Eibach T and Völkel G, Optimising Gröbner bases on Bivium, *Mathematics in Computer Science* 2010, **3**(2): 159–172.

[32] Huang Z and Lin D, A new method for solving polynomial systems with noise over $\mathbb{F}_{22}$ and its applications in cold boot key recovery, *Selected Areas in Cryptography,* LNCS 7707, Windsor, Canada, 2013, 16–33.

[33] Faugère J C and Ars G, An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases, TR No. 4739, INRIA, 2003.

[34] Gao X S and Huang Z, Characteristic set algorithms for equation solving in finite fields, *Journal of Symbolic Computation,* 2012, **47**(6): 655–679.

[35] Buchberger B, A criterion for detecting unnecessary reductions in the construction of Gröbner basis, *Proceedings of EUROSAM'79, Lect. Notes in Comp. Sci.*, Springer, Berlin, 1979, **72**: 3–21.

[36] Eder C, An analysis of inhomogeneous signature-based Gröbner basis computations, *J. Symb. Comput.,* 2013, **59**: 21–35.

[37] Gao S H, Volny F, and Wang M S, A new algorithm for computing Gröbner bases, *Cryptology ePrint Archive*, Report 2010/641, 2010.