An Efficient Variant of F4 Algorithm for Solving MQ Problem

Kosuke Sakata[1] and Tsuyoshi Takagi[1]

Department of Mathematical Informatics, The University of Tokyo, Japan
\{kosuke-sakata-rb,takagi\}@g.ecc.u-tokyo.ac.jp

Abstract. In this paper, we propose an enhanced variant of the F4 algorithm specifically designed for efficiently solving multivariate quadratic (MQ) problems, which are central to many post-quantum cryptographic schemes. Our approach overcomes a major inefficiency of conventional F4 by integrating a Hilbert-driven strategy that determines the optimal number of S-polynomials to generate at each degree, thereby reducing unnecessary zero reductions. We further introduce refined pair selection techniques that prioritize candidates yielding S-polynomials with smaller leading terms, which in turn minimizes the dimensions of intermediate matrices used during reduction. Experimental results show that our implementation outperforms state-of-the-art systems such as M4GB and Magma's F4 in both single-core and multi-core environments. Notably, our method sets new records in the Fukuoka MQ Challenge for Type VI problems over $\mathbb{F}_{31}$ with $m = 21, 22, 23, 24$ demonstrating the robustness and practical impact of our approach in solving highly challenging MQ instances. According to the computational complexity estimation formula given in [24], the problem with $m = 24$ is approximately 47,627 times harder than the previous record case with $m = 20$.

Keywords: post-quantum cryptography · F4 · MQ problem

1 Introduction

Public key cryptography is an essential technology for maintaining the security of information and communication systems. Currently, RSA and ECC are employed. However, with the practical realization of quantum computers, there is concern regarding the decreased security of recent public key cryptography. Shor's algorithm [19] has the potential to break current public key encryption schemes in polynomial time. Consequently, research is underway to develop post-quantum cryptography, capable of withstanding quantum computers.

The National Institute of Standards and Technology (NIST) is currently selecting post-quantum cryptographic schemes. Evaluating the difficulty of breaking these candidates is an important task. Currently, a selection process for additional digital signatures is also underway, and multivariate polynomial cryptography is an important candidate. Multivariate polynomial cryptography relies on the security of the multivariate quadratic (MQ) problem.

Gröbner basis computation is an efficient method for computing MQ problems. Moreover, improving Gröbner basis algorithms is closely related to the evaluation of the computational complexity of MQ problems. Therefore, improvement of the algorithms is an exceptionally important research topic in computational algebra and cryptography.

There are several algorithms for solving MQ problems. For example, there are the XL-related algorithms [25, 26, 27, 28, 29], F4 [9], F5 [10], and so on. Among them, F4 has produced many of the best results in the Fukuoka MQ Challenge [24] (a competition for

solving MQ problems). In particular, the current records for Types V and VI have been achieved by implementations derived from F4, underscoring its practical effectiveness for real instances [24]. Therefore, in this paper we develop methods to accelerate F4 for solving MQ problems and evaluate the computational complexity of the resulting algorithm.

A common issue in F4 is the excessive calculation of zero reductions, which are irrelevant to the output. The Hilbert-driven algorithm [22] uses the Hilbert series, then can avoid many zero reductions. In the Hilbert-driven algorithm, after computing an S-polynomial with degree d, the Hilbert function is computed to determine whether the subsequently calculated S-polynomials of degree d would result in zero reductions.

In this paper, we focus on enhancing the efficiency of Gröbner basis computations, specifically targeting multivariate quadratic problems. Our main contribution is an improved variant of the F4 algorithm that is particularly well-suited for solving MQ problems. To achieve significant performance gains, our approach introduces several techniques to mitigate the inefficiencies of the conventional F4 algorithm: the excessive computation of zero reductions and large matrix size.

We incorporate a Hilbert-driven strategy that uses the Hilbert series to determine the necessary number of S-polynomials at each degree. This method enables us to restrict the generation of S-polynomials to those that are most likely to contribute to the final Gröbner basis, thereby reducing redundant calculations. In addition, we propose refined pair selection strategies–namely, `SelectPairLimited`, and `SelectPairM`–which prioritize pairs that yield S-polynomials with smaller leading terms and similar structural properties. These strategies further reduce the computational workload and memory requirements by limiting the size of the intermediate matrices during the reduction process.

Our experimental results demonstrate that our algorithm outperforms existing implementations, such as M4GB and Magma's F4, in both single-core and multi-core environments. Notably, our approach sets new records in the Fukuoka MQ Challenge for Type VI problems over $\mathbb{F}_{31}$ with $m = 21, 22, 23, 24$. Although our current high-performance implementation targets $\mathbb{F}_{31}$, the algorithm itself is field-independent and can be applied to systems over other prime fields and extension fields such as $\mathbb{F}_{2^8}$, once corresponding arithmetic kernels are available.

Multivariate quadratic (MQ) systems appear at the core of many post-quantum signature schemes, such as UOV [5] and its variants, where signing reduces to solving a dense homogeneous quadratic system after fixing the vinegar variables. Our algorithm targets exactly this dense regime and thus directly impacts the cost of Gröbner-basis attacks on such schemes.

The novelty of this work lies in the algorithmic integration rather than in any single low-level optimization. More precisely, we introduce an MQ-oriented F4 variant in which the degree-increasing phase is explicitly controlled by LGB-based Hilbert predictions, the degree-decreasing phase is handled by a separate restricted-generation strategy, and both phases are designed to reduce the size of the intermediate matrices. This algorithmic design is then realized through a dense linear-algebra implementation over $\mathbb{F}_{31}$, yielding substantial practical gains.

MQ systems can be broadly divided according to the ratio m/n. When $m/n \approx 1$, which typically arises after guessing in hybrid attacks, the complexity of Gröbner-basis algorithms is largely governed by the degree of regularity and dominated by dense linear algebra. In contrast, when $m/n \approx a > 1$, the degree of regularity is usually lower and alternative approaches such as linearization may become competitive. Our work mainly targets the former regime, although the proposed strategy may also be applicable more broadly.

Our Contributions. The main contributions of this paper are as follows: (1) We integrate a Hilbert-driven selection mechanism (HDF4) into F4, theoretically bounded by the predicted basis distribution from the LGB algorithm. (2) We propose a novel pair se-

lection strategy (`SelectPairLimited` and `SelectPairM`) minimizing matrix size. (3) We perform an extensive experimental evaluation on large multivariate quadratic (MQ) instances. Our implementation not only outperforms state-of-the-art systems such as M4GB and Magma's F4 in both single-core and multi-core environments, but also achieves new records in the Fukuoka MQ Challenge (Type VI over $\mathbb{F}_{31}$ with $m = 21, 22, 23, 24$), thereby confirming the practical effectiveness of our algorithm on real benchmark problems. According to the computational complexity estimation formula given in [24], the problem with $m = 24$ is approximately 47,627 times harder than the previous record case with $m = 20$. This clearly demonstrates the significant computational leap achieved in this work.

The remainder of the paper is organized as follows. In Section 2, we lay out the necessary preliminaries, including key definitions and notational conventions used throughout the work. Section 3 describes F4 algorithm in detail. Section 4 is about an algorithm LGB for computing the number of elements of minimal Gröbner basis for generic sequences. In Section 5, we introduce our algorithm, a refined variant of F4 optimized for MQ problems. Section 6 is about implementation of the proposed algorithm. More than 90% of the total running time is spent in matrix operations, we focus on their implementations. Section 7 provides a comprehensive comparison of the computation times and memory usage between our implementation and existing algorithms. In Section 8, we report our new records in the Fukuoka MQ Challenge, demonstrating the effectiveness of our approach on challenging MQ instances.

2 Preliminaries

2.1 Notation

We introduce the notations and definitions that form the foundation for the discussions in this paper. We denote by $\mathbb{Z}$ the set of integers, and by $\mathbb{Z}_{\geq 0}$ the set of non-negative integers. Let k be an arbitrary field, and for any positive integer q, let $\mathbb{F}_q$ denote the finite field with q elements.

Consider the polynomial ring $R = k[x_1, x_2, \ldots, x_n]$, which is the ring of polynomials in n variables over the field k. A monomial in R is any product of the form $x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$, where the exponents $a_1, a_2, \ldots, a_n$ belong to $\mathbb{Z}_{\geq 0}$. For any two monomials t and u in R, we write $t \mid u$ to indicate that t divides u.

A monomial ordering is an indispensable tool in the study of Gröbner bases. Throughout the paper, monomials are ordered by the degree reverse lexicographic order with $x_1 > x_2 > \cdots > x_n$, i.e., monomials are first compared by total degree, and among monomials of the same degree, by reverse lexicographic order induced by this variable ordering. We denote such an ordering by $<$ and specifically adopt the degree reverse lexicographic order. For a given non-negative integer d, we define M_d as the set of all monomials in R of degree d. The cardinality of any set A is written as $\#A$.

Let $f \in R$ be a nonzero polynomial. The *leading term* of f, denoted by $\mathrm{LT}(f)$, is defined as the largest term appearing in f with respect to the selected monomial ordering. The coefficient corresponding to $\mathrm{LT}(f)$ is denoted by $\mathrm{LC}(f)$, $\mathrm{LM}(f)$ is the monomial part of $\mathrm{LT}(f)$, and the total degree of f is denoted by $\deg(f)$.

For a collection of polynomials $F = \{f_1, f_2, \ldots, f_m\}$, the ideal generated by F is denoted by $\langle F \rangle$. We denote the set of leading terms of the elements of F by $LT(F)$ and the ideal generated by these leading terms by $\langle \mathrm{LT}(F) \rangle$. Furthermore, if $I \subset R$ is an ideal and G is a set of polynomials generating I, then G is called a Gröbner basis for I if, for every nonzero polynomial $f \in I$, there exists some $g \in G$ such that $\mathrm{LT}(g)$ divides $\mathrm{LT}(f)$.

S-polynomial $\mathrm{Spoly}(f, g)$ for a pair of polynomials $f, g \in R$ is defined as

$$\mathrm{Spoly}(f, g) = \frac{\mathrm{lcm}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(f)} f - \frac{\mathrm{lcm}(\mathrm{LT}(f), \mathrm{LT}(g))}{\mathrm{LT}(g)} g$$

where lcm represents a function that calculates the least common multiple of monomials. With $f, h \in R$ such that $\mathrm{LT}(h) \mid \mathrm{LT}(f)$, the following calculation is called a reduction:

$$f - \frac{\mathrm{LT}(f)}{\mathrm{LT}(h)} h$$

In this paper, we call the computation of reducing an S-polynomial to zero as "zero reduction". Zero reductions do not contribute new elements to the Gröbner basis, and thus are considered computationally wasteful. Gröbner basis algorithms can be more efficient by avoiding zero reductions. Several methods for preventing zero reductions have been proposed in prior works such as [13] and [10].

In the remainder of this paper, we denote by $\mathrm{MQ}(n, m)$ an MQ problem consisting of n variables and m polynomials.

2.2 Hilbert Series

We introduce a definition of the Hilbert series. Let k be a field and $d \geq 0$ be an integer. Let R_d denote the degree d homogeneous component of R. We assume that the ideal $I \subset R$ is a homogeneous ideal. For an ideal $I \subset R$, we define I_d as

$$I_d = \{f \in I \mid f \text{ is a homogeneous polynomial with } deg(f) = d\} \cup \{0\}.$$

Definition 1 ([18])**.** We define the Hilbert function of the quotient ring R/I as follows:

$$\begin{array}{rccc} h_{R/I}\colon & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & d & \longmapsto & \dim_k (R/I)_d \end{array}$$

Using the function $h_{R/I}$, we define the Hilbert series $H_{R/I}(z)$ of the quotient ring R/I as follows:

$$H_{R/I}(z) = \sum_{d \geq 0} h_{R/I}(d) z^d.$$

2.3 Generic Sequence

If a polynomial sequence is chosen randomly, it is likely that the sequence holds a generic property. A polynomial sequence is called generic if it satisfies certain generic properties with respect to the coefficients [17].

In this section, the features of generic homogeneous sequences are summarized. Let k be an infinite field and R be a polynomial ring over k.

Definition 2 ([17, 18])**.** A monomial ideal $I \subset R$ is called a *weakly reverse lexicographic ideal* if for any minimal generator $m \in I$ (i.e., a monomial in the minimal generating set of I that is not divisible by any other generator of I), whenever $m' > m$ and $\deg(m') = \deg(m)$, we have $m' \in I$.

The following conjecture is known as the Moreno-Socías conjecture.

Conjecture 1 ([12, 18])**.** *Let F be a generic homogeneous sequence, and let $I = \langle F \rangle$. Then, $\langle LT(I) \rangle$ forms a weakly reverse lexicographic ideal.*

Definition 3. Let $\sum_{i=0}^{\infty} a_i z^i$ be a formal power series with $a_i \in \mathbb{Z}$. We denote by $\left[\sum_{i=0}^{\infty} a_i z^i\right]$ the series obtained by truncating $\sum_{i=0}^{\infty} a_i z^i$ at the first index i for which $a_i \leq 0$.

The next conjecture is called the Fröberg conjecture.

Conjecture 2. *[17, 18] Let $F = \{f_1, ..., f_m\}$ be a generic homogeneous sequence with $\deg(f_i) = d_i$, and let $I = \langle F \rangle$. Then,*

$$H_{R/\langle f_1,\ldots,f_m\rangle}(z) = \left[\frac{\prod_{i=1}^{m}(1-z^{d_i})}{(1-z)^n}\right].$$

Definition 4 ([18])**.** Let $I \subset R$ be homogeneous and $f \in R$ be homogeneous of degree d. We say that f is *semi-regular* on R/I if for every e the multiplication map $(R/I)_{e-d} \xrightarrow{\cdot f} (R/I)_e$ has maximal rank. A sequence $(f_1, \ldots, f_m)$ is *semi-regular* if f_i is semi-regular on $R/\langle f_1, \ldots, f_{i-1}\rangle$ for all i.

Let F be a homogeneous sequence. We define that F is a semi-regular sequence if F^{top} is a semi-regular sequence.

Conjecture 3 ([18])**.** *A generic homogeneous sequence is a homogeneous semi-regular sequence.*

According to [18], if the Moreno-Socías conjecture holds, the Fröberg conjecture and Conjecture 3 also hold.

Theorem 1 ([18])**.** *A sequence of homogeneous polynomials $\{f_1, \ldots, f_m\}$ with $\deg(f_i) = d_i$ is a semi-regular sequence if and only if for every $1 \le j \le m$ the following holds:*

$$H_{R/\langle f_1,\ldots,f_j\rangle}(z) = \left[\frac{\prod_{i=1}^{j}(1-z^{d_i})}{(1-z)^n}\right] \quad \textit{for } 1 \le j \le m.$$

3 Overview of F4 algorithm

In this section, we review the F4 algorithm, which was originally introduced by Faugère [9]. The version described here follows the formulation in Chapter 10, §3 of [8] and serves as the basis for our proposed variant presented in Section 5. The way the auxiliary polynomial set $\widetilde{G}$ is used for reduction and the manner in which the matrices are constructed differ from those in [8]. Therefore, we provide a detailed explanation here to clarify the distinctions between the formulation in [8] and our proposed algorithm in Section 5.

3.1 F4

In Algorithm 1, the input is a set of m polynomials $F = \{f_1, \ldots, f_m\}$, and the output is a Gröbner basis G of the ideal $I = \langle F \rangle$. First, the polynomial set F is directly assigned to G, and an auxiliary polynomial set $\widetilde{G}$ is also initialized as F. The set $\widetilde{G}$ is used later for reductions of S-polynomials.

Next, a set of all pairs $B = \{(i, j) \mid 1 \le i < j \le \#G\}$ is initialized; each pair corresponds to two polynomials from which an S-polynomial will be generated. The algorithm then iterates as long as B is nonempty. In each iteration, it first determines the minimal degree d among all S-polynomials that can be generated from the pairs in B. Thereafter, using the function `SelectPair`, the set B is partitioned into a candidate subset B' and the remaining pair set B. For each pair (i, j) in B', the S-polynomial $\mathrm{Spoly}(f_i, f_j)$ is computed, and these S-polynomials are collected into the set L_S. Then, using L_S and the auxiliary set $\widetilde{G}$, the function `ComputeP` is invoked to generate a matrix P and a set M_B of monomials that are not reducible by the leading terms in $\widetilde{G}$.

The matrix P is then processed by the function `RowEchelonForm`. From the resulting echelon matrix Q, a corresponding polynomial is extracted from each row to form the

Algorithm 1 `F4`

Require: $F = \{f_1, ..., f_m\} \subset R$
Ensure: a Gröbner Basis $G \subset R$ for $I = \langle F \rangle$

$G \leftarrow F$
$\widetilde{G} \leftarrow F$
$B \leftarrow \{(i,j) \mid 1 \leq i < j \leq \#G\}$
while $B \neq \varnothing$ **do**
  $d \leftarrow$ the minimal degree in $\{\deg((i,j)) \mid (i,j) \in B\}$
  $B', B \leftarrow$ `SelectPair`(B, d)
  $L_S \leftarrow \{\mathrm{Spoly}(f_i, f_j) \mid (i,j) \in B'\}$
  $P, M_B \leftarrow$ `ComputeP`$(L_S, \widetilde{G})$
  $Q \leftarrow$ `RowEchelonForm`(P)
  $H \leftarrow$ polynomial form of Q
  for $h \in H$ **do**
    if $\mathrm{LM}(h) \in M_B$ **then**
      $G \leftarrow G \cup \{h\}$
      $B \leftarrow B \cup \{(i, \#G) \mid 1 \leq i < \#G\}$
    end if
    if $h' \in \widetilde{G}$ such that $\mathrm{LM}(h') = \mathrm{LM}(h)$ **then**
      $\widetilde{G} \leftarrow \widetilde{G} \setminus \{h'\}$
      $\widetilde{G} \leftarrow \widetilde{G} \cup \{h\}$
    else
      $\widetilde{G} \leftarrow \widetilde{G} \cup \{h\}$
    end if
  end for
end while
return $\{G\}$

polynomial set H. For each polynomial h in H, if the leading monomial of h is contained in M_B, then h is regarded as a new polynomial and is added to G. With each addition to G, new pairs are generated and the set B is updated accordingly. In addition, the new polynomials are also incorporated into the auxiliary set $\widetilde{G}$ so that more reduced forms of the polynomials can be used in the subsequent computations.

Through this iterative process, the final set G that is produced becomes the Gröbner basis for the input ideal $I = \langle F \rangle$, and it is then output.

3.2 SelectPair

We describe the pair selection method `SelectPair` used in the F4 for Gröbner basis computation. This function is designed to extract, from the candidate set of polynomial pairs B, those pairs that contribute to the generation of S-polynomials. The selection method introduced here computes S-polynomials in order of increasing degree, an approach referred to as the *normal selection strategy*.

First, for each pair (i,j) in the given candidate set B, the degree of the S-polynomial to be generated is evaluated. The degree of each pair is determined by the degree of the least common multiple of the leading monomials of the corresponding polynomials f_i and f_j, i.e.,

$$\deg((i,j)) = \deg(\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))). \qquad (\#)$$

To focus only on the pairs with the minimum degree d, we extract from B the subset

Algorithm 2 `SelectPair`

Require: $B \subset \mathbb{N}^2, d \in \mathbb{N}$
Ensure: $B' \subset \mathbb{N}^2, B \subset \mathbb{N}^2$
1: $B' \leftarrow \varnothing$
2: $B'' \leftarrow \{b \in B \mid \deg((i,j)) = d\}$
3: $B \leftarrow B \setminus B''$
4: **while** $B'' \neq \varnothing$ **do**
5: $(i,j) \leftarrow$ an element in B''
6: **if** `Criterion`$((i,j), B') =$ `false` **then**
7: $B' \leftarrow B' \cup \{(i,j)\}$
8: **end if**
9: $B'' \leftarrow B'' \setminus \{(i,j)\}$
10: **end while**
11: **return** $\{B', B\}$

$B'' = \{\, b \in B \mid \deg(b) = d \,\}$. Then, we remove B'' from the candidate set B.

Next, for each pair in B'', the function `Criterion` is used to evaluate the pair. The condition `Criterion` is defined in section 3.4. This evaluation determines whether the corresponding S-polynomial is liable to a zero reduction. If the evaluation returns `false`, the pair is judged to be a promising candidate for computation, and it is added to the selection set B'. This process is repeated until all pairs in B'' have been processed (each pair is evaluated once and then removed from B'').

Finally, `SelectPair` returns the selected pair set B' together with the updated candidate set B.

3.3 ComputeP

We describe the function `ComputeP` used in the F4 to efficiently perform the reduction of S-polynomials. This function takes as input the set of S-polynomials L_S and the auxiliary set $\widetilde{G}$, and we obtain the polynomial set L_R required for the reduction, as well as the set of monomials M_A that are divisible by the leading terms of $\widetilde{G}$ and the set M_B that are not. Then, using L_S, L_R, M_A, and M_B, the matrix P is constructed as described in Section 3.5.

First, all monomials appearing in the polynomials of L_S are collected into a set M. Initially, both M_A and M_B are set to be empty. Here, M_A is defined as the set of monomials that are divisible by the leading terms of $\widetilde{G}$, while the monomials that are not divisible by the leading terms of $\widetilde{G}$ are assigned to M_B.

According to the monomial ordering, the largest monomial r in $M \setminus (M_A \cup M_B)$ is extracted. If there exists a polynomial $\widetilde{f} \in \widetilde{G}$ such that $\mathrm{LM}(\widetilde{f})$ divides r, then among all such polynomials, we choose $\widetilde{f}'$ whose leading monomial $\mathrm{LM}(\widetilde{f}')$ is maximal with respect to the monomial ordering. A monomial s is then computed such that $\mathrm{LM}(s \cdot \widetilde{f}') = r$. The monomials appearing in $s \cdot \widetilde{f}'$ are added to M, and the polynomial $s \cdot \widetilde{f}'$ is stored in L_R. In this case, r is added to M_A. If no such $\widetilde{f}$ exists, then r is added to M_B.

The reason for choosing $\widetilde{f}'$ with the largest leading monomial is that it minimizes the monomial order of the multiplier monomial s used for reduction. As a result, it is expected to reduce the number of times the reduction step must be applied, thereby improving overall efficiency.

This loop is repeated until there remain no unprocessed monomials in M (i.e., until $M = M_A \cup M_B$). Finally, using the information in M_A, M_B, L_R, and L_S, the matrix P

Algorithm 3 `ComputeP`

Require: $L_S \subset R, \widetilde{G} \subset R$
Ensure: $P \in \mathbb{F}_q^{(\#L_R+\#L_S)\times(\#M_A+\#M_B)}, M_B$
1: $M \leftarrow \mathrm{Mon}(L_S)$
2: $M_A \leftarrow \varnothing$
3: $M_B \leftarrow \varnothing$
4: $L_R \leftarrow \varnothing$
5: **while** $M \neq M_A \cup M_B$ **do**
6: $r \leftarrow$ the largest monomial in $M \setminus (M_A \cup M_B)$
7: **if** there exists $\widetilde{f} \in \widetilde{G}$ such that $\mathrm{LM}(\widetilde{f}) \mid r$ **then**
8: $\widetilde{f'} \leftarrow$ Choose $\widetilde{f} \in \widetilde{G}$ such that $\mathrm{LM}(\widetilde{f}) \mid r$ where $\mathrm{LM}(\widetilde{f})$ is maximal among all such $\widetilde{f}$
9: $s \leftarrow \frac{r}{\mathrm{LM}(\widetilde{f'})}$
10: $M \leftarrow M \cup \mathrm{Mon}(s \cdot \widetilde{f'})$
11: $L_R \leftarrow L_R \cup \{s \cdot \widetilde{f'}\}$
12: $M_A \leftarrow M_A \cup \{r\}$
13: **else**
14: $M_B \leftarrow M_B \cup \{r\}$
15: **end if**
16: **end while**
17: $P \leftarrow$ matrix shown in section 3.5 using M_A, M_B, L_R, L_S
18: **return** $\{P, M_B\}$

is constructed as explained in section 3.5. This matrix P embeds the coefficients of the components from the S-polynomial set L_S and the reduction set L_R, and serves as the foundation for the subsequent row-reduction process via `RowEchelonForm`.

Let $\widetilde{f} \in \widetilde{G}$ be the polynomial such that $\mathrm{LM}(\widetilde{f}) \in r$ and $\mathrm{LM}(\widetilde{f})$ is the largest among such polynomials with respect to the monomial order.

Choose $\widetilde{f} \in \widetilde{G}$ satisfying $\mathrm{LM}(\widetilde{f}) \in r$, where $\mathrm{LM}(\widetilde{f})$ is maximal among all such $\widetilde{f}$.

3.4 Criterion

We describe the evaluation criterion, called `Criterion`, that is used to select the polynomial pairs for which S-polynomials are generated. This function is applied to a given polynomial pair (f_i, f_j) to determine whether the resulting S-polynomial will eventually reduce to zero.

First, if the least common multiple (lcm) of the leading monomials $\mathrm{LM}(f_i)$ and $\mathrm{LM}(f_j)$ equals the product $\mathrm{LM}(f_i) \cdot \mathrm{LM}(f_j)$, then the S-polynomial is subject to zero reduction, and therefore `Criterion` returns `true`. Moreover, if there exists a polynomial f_k in the already selected pair set B' such that its leading term $\mathrm{LM}(f_k)$ divides the lcm of $\mathrm{LM}(f_i)$ and $\mathrm{LM}(f_j)$, then it is similarly determined that the S-polynomial generated from f_i and f_j is unnecessary; in this case, `Criterion` also returns `true`.

Only if neither of these conditions holds is the pair considered to be a useful candidate for generating an S-polynomial, and `Criterion` returns `false`. In this way, the evaluation mechanism prevents the generation of unnecessary S-polynomials.

Algorithm 4 `Criterion`

Require: $(i, j) \in \mathbb{N}^2, B' \subset \mathbb{N}^2$
Ensure: `true` or `false`
1: **if** $\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j)) = \mathrm{LM}(f_i)\mathrm{LM}(f_j)$ **then**
2: **return** `true`
3: **end if**
4: **if** there exists $(i, k) \in B'$ and $(j, k) \in B'$ and $\mathrm{LM}(f_k) \mid \mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))$ **then**
5: **return** `true`
6: **end if**
7: **return** `false`

$$
\begin{cases}
r_1 = \sum_{i=1}^{l_m} a_{1,i} m_i + \sum_{i=1}^{l_n} b_{1,i} n_i \\
\quad \vdots \\
r_{l_r} = \sum_{i=1}^{l_m} a_{l_r,i} m_i + \sum_{i=1}^{l_n} b_{l_r,i} n_i \\
s_1 = \sum_{i=1}^{l_m} c_{1,i} m_i + \sum_{i=1}^{l_n} d_{1,i} n_i \\
\quad \vdots \\
s_{l_s} = \sum_{i=1}^{l_m} c_{l_s,i} m_i + \sum_{i=1}^{l_n} d_{l_s,i} n_i
\end{cases}
\longrightarrow
\begin{array}{c}
\\
r_1 \\
\vdots \\
r_{l_r} \\
s_1 \\
\vdots \\
s_{l_s}
\end{array}
\begin{array}{c}
\begin{array}{cccccc} m_1 & \cdots & m_{l_m} & n_1 & \cdots & n_{l_n} \end{array} \\
\begin{bmatrix}
a_{1,1} & \cdots & a_{1,l_m} & b_{1,1} & \cdots & b_{1,l_n} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
a_{l_r,1} & \cdots & a_{l_r,l_m} & b_{l_r,1} & \cdots & b_{l_r,l_n} \\
c_{1,1} & \cdots & c_{1,l_m} & d_{1,1} & \cdots & d_{1,l_n} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
c_{l_s,1} & \cdots & c_{l_s,l_m} & d_{l_s,1} & \cdots & d_{l_s,l_n}
\end{bmatrix}
\end{array}
= P
$$

Figure 1: Diagram illustrating the construction of the matrix P from the polynomial sets $L_R = \{r_1, \ldots, r_{l_r}\}$ and $L_S = \{s_1, \ldots, s_{l_s}\}$, and the monomial sets $M_A = \{m_1, \ldots, m_{l_m}\}$ and $M_B = \{n_1, \ldots, n_{l_n}\}$.

3.5 Generation of the Matrix P

We explain the construction of the matrix P produced by the `ComputeP` function. Let $L_S = \{s_1, s_2, \ldots, s_{l_s}\}$ be the set of S-polynomials and $L_R = \{r_1, r_2, \ldots, r_{l_r}\}$ be the set of polynomials used for the reduction. Also, let $M_A = \{m_1, m_2, \ldots, m_{l_m}\}$ and $M_B = \{n_1, n_2, \ldots, n_{l_n}\}$ be the sets of monomials obtained at the time of line 17 in `ComputeP`. Furthermore, assume that the polynomials in L_R are arranged in descending order with respect to their leading monomials (i.e., for each i, $\mathrm{LM}(r_i) > \mathrm{LM}(r_{i+1})$). Similarly, the sets M_A and M_B are arranged in descending order.

Denote the coefficients of the monomials in the polynomial r_u (for $1 \le u \le l_r$) and in the polynomial s_v (for $1 \le v \le l_s$) as follows:

$$
r_u = \sum_{i=1}^{l_m} a_{u,i}\, m_i \; + \; \sum_{i=1}^{l_n} b_{u,i}\, n_i,
$$

$$s_v = \sum_{i=1}^{l_m} c_{v,i}\, m_i \;+\; \sum_{i=1}^{l_n} d_{v,i}\, n_i.$$

Then, from the polynomials $\{r_1, r_2, \ldots, r_{l_r},\ s_1, s_2, \ldots, s_{l_s}\}$, the matrix P is constructed as illustrated in Figure 1. In this construction, the upper block of rows corresponds to the polynomials in L_R and the lower block corresponds to those in L_S. Each column represents a monomial from the set $M_A \cup M_B$ and each entry in the matrix is the coefficient of the corresponding monomial in the given polynomials.

Let us denote by $A \in k^{l_r \times l_m}$, $B \in k^{l_r \times l_n}$, $C \in k^{l_s \times l_m}$, and $D \in k^{l_s \times l_n}$ the submatrices containing the coefficients $a_{*,*}$, $b_{*,*}$, $c_{*,*}$, and $d_{*,*}$, respectively. Then, the matrix P can be written in block form as

$$P = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Moreover, the submatrix A forms an upper triangular matrix due to the ordering of L_R and M_A.

In Algorithm 1, there is a procedure (at line 10) that converts the rows of the matrix P back into polynomials, which is essentially the inverse operation of the construction illustrated in Figure 1.

3.6 Computation of the Row Echelon Form

In this section, we demonstrate a method to efficiently compute the row echelon form of the matrix P by taking advantage of the fact that the matrix A is already upper triangular. The process for obtaining the row echelon form of P is illustrated in Figure 2.

In Figure 2, for convenience a diagram is shown for the case in which no S-polynomial leads to a zero reduction. However, the computation process is similar even in cases where zero reduction occurs; if a zero reduction appears, a row consisting entirely of zeros will occur at the bottom of the matrix after executing the T_{CB} or T_R operations. In the diagram, the top matrix represents P and the bottom matrix is the row echelon form of P.

The following operations in Figure 2 are explained:

T_{AB}: Since A is already an upper triangular matrix, each row contains a pivot. Therefore, by using these pivots, A can be transformed into the identity matrix without any rank loss. By performing this transformation on the A part of the augmented matrix $[A \,|\, B]$, the B block is converted into B'.

T_{CB}: Using the matrix $[A \,|\, B']$, the C block in the augmented matrix $[C \,|\, D]$ is eliminated. The modified D after this elimination is denoted as D'. This operation is achieved by computing the product

$$D' \leftarrow D - C\,B'.$$

T_R: The row echelon form of D' is then computed. As a result, the portion that becomes the identity matrix is denoted by E, and the remaining portion is denoted by D''.

T_{BD}: Using the matrix $[E \,|\, D'']$, the B'_1 part of $B' = [B'_1 \mid B'_2]$ is eliminated. The resulting modified block B'_2 is denoted as B''. This elimination is carried out by computing the product

$$B'' \leftarrow B'_2 - B'_1\,D''.$$

As a result of these matrix operations, the row echelon form of P is obtained. Since the matrix operations account for the majority of the computational load in Gröbner basis calculations, it is crucial to accelerate the computation of `RowEchelonForm`–that is, to implement the matrix products (T_{CB} and T_{BD}) and the computation of the echelon form (T_R) as efficiently as possible.

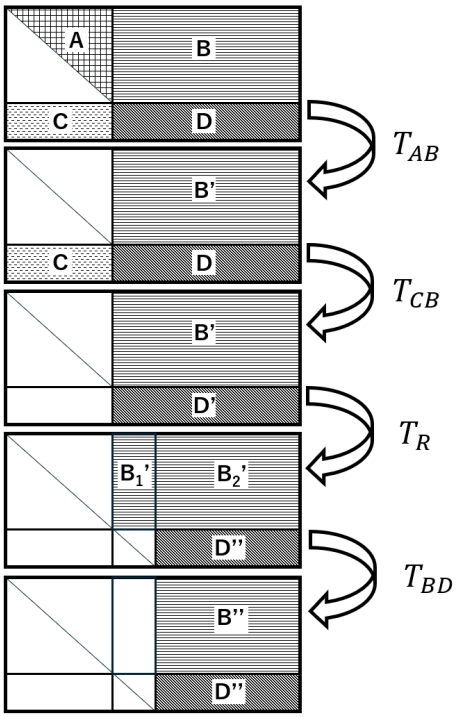

Figure 2: Process of row echelon form of the matrix P

Remark 1. This remark explains the difference between the F4 introduced in this paper and signature-based algorithms, such as F5. In signature-based algorithms, each polynomial used in the computation is augmented with information called a signature. An ordering is imposed on these signatures, and it is not permitted to reduce a polynomial with a smaller signature by one with a larger signature. In terms of matrix operations, this means that when one row is used to reduce another, the reduction can only be carried out in one direction. In other words, the matrix operations in signature-based algorithms cannot yield the final matrix form as shown in Figure 2. F4 introduced in this paper uses the sufficiently simplified polynomials shown in the final matrix of Figure 2 to generate S-polynomials (G) and reduce S-polynomials ($\widetilde{G}$) of the next degree. This is the difference between F4 and signature-based algorithms.

4 LGB Algorithm

We present the algorithm `LGB` [20], which computes the number of elements N_d in the minimal Gröbner basis at each degree d for a generic sequence $F = \{f_1, f_2, \ldots, f_m\}$, characterized by the number of variables n, the number of polynomials m, and the degrees $\{d_i\}_{i=1}^m$ of the polynomials. This approach is based on the assumption of the Moreno-Socías conjecture (and the Fröberg conjecture) and utilizes the Hilbert series. The correctness and termination of the `LGB`, as well as a more efficient version of it, are presented in [20]. We describe a simple version of `LGB`.

First, for the sequence F, we compute the Hilbert series $H_{R/\langle F\rangle}(z)$ defined as

$$H_{R/\langle F\rangle}(z) = \frac{\prod_{i=1}^m (1 - z^{d_i})}{(1-z)^n}.$$

Algorithm 5 `LGB`

Require: $n \in \mathbb{N}, m \in \mathbb{N}, \{d_i\}_{i=1}^{m} \subset \mathbb{N}$
Ensure: $\{N_d\}_{d=1}^{D} \subset \mathbb{N}$ of a generic homogeneous sequence $F = \{f_1, ..., f_m\}$ with n variables, m polynomials and $\deg(f_i) = d_i$

1: $H_{R/\langle F\rangle}(z) \leftarrow \left[\frac{\prod_{i=1}^{m}(1-z^{d_i})}{(1-z)^n}\right]$
2: $L_G \leftarrow \varnothing$
3: $d \leftarrow 0$
4: **while** $H_{R/\langle F\rangle}(z) \neq H_{R/\langle L_G\rangle}(z)$ **do**
5: $\quad B_d \leftarrow \{m \in M_d \mid m \notin \langle L_G\rangle\}$
6: $\quad N_d \leftarrow \#B_d - h_{R/\langle F\rangle}(d)$
7: $\quad L_G \leftarrow L_G \cup \{\text{the } N_d \text{ monomials in } B_d \text{ that are largest}\}$
8: $\quad d \leftarrow d + 1$
9: **end while**
10: **return** $\{N_d\}_{d=1}^{D}$

This series provides the coefficient $h_{R/\langle F\rangle}(d)$ for each degree d, which encodes the dimension of $R/\langle F\rangle$ in that degree.

`LGB` begins by initializing the set L_G of leading monomials of the Gröbner basis to be empty and setting the degree counter d to 0. Then, in each iteration, the algorithm considers the set M_d of all monomials of degree d and extracts the subset $B_d = \{m \in M_d \mid m \notin \langle L_G\rangle\}$, which contains all candidate leading monomials for the minimal Gröbner basis at degree d.

Next, the number of minimal Gröbner basis elements N_d at degree d is determined by subtracting $h_{R/\langle F\rangle}(d)$ from the total number of elements in B_d. Furthermore, the monomials in B_d are arranged in descending order with respect to the chosen monomial order, and the top N_d monomials are selected and added to L_G. At that point, L_G represents the set of leading terms of the minimal Gröbner basis at degree d. In the subsequent iterations, the information from all previously obtained leading monomials (up to degree d) is incorporated into the computation.

This loop continues until the Hilbert series of $R/\langle F\rangle$ equals the Hilbert series of $R/\langle L_G\rangle$; that is, $H_{R/\langle F\rangle}(z) = H_{R/\langle L_G\rangle}(z)$. At that point, the final set L_G constitutes the complete set of leading terms of the Gröbner basis for the input sequence F, and the collection $\{N_d\}_{d=1}^{D}$ of minimal Gröbner basis element counts at each degree is obtained. Here, D is defined as the largest d for which $N_d \neq 0$.

Since this algorithm relies solely on monomial divisibility checks, it has a lower computational cost and operates faster in practice compared to Gröbner basis computations that primarily use matrix operations. Although the LGB algorithm presented here is sufficiently fast for practical purposes, a more efficient version is also described in [20].

5 Proposal of an algorithm for efficiently computing MQ problems

In this section, we propose an algorithm for the efficient computation of MQ problems. In the Gröbner basis computation for MQ(n, m) where $n < m$, there are two distinct computational phases: one in which the degrees of the S-polynomials increase and one in which the degrees decrease. The phase where the degrees of the S-polynomials increase is computed using the algorithm `HDF4`, and the phase where the degrees decrease is computed

using the algorithm MF4. Although both HDF4 and MF4 are based on the F4, they differ from F4 in the way the S-polynomial pairs are selected. In HDF4, the method for selecting S-polynomials employs the Hilbert series to limit the number of S-polynomials computed, which is similar to the approach used in Hilbert-driven algorithms. In contrast, the S-polynomial selection method in MF4 is an improved version of the pair selection method employed in the MXL algorithm [16].

The algorithm proposed in this paper is designed to solve MQ problems in which the number of variables n is less than the number of equations m (i.e., $n < m$). This is based on the assumption that for MQ problems with $n \geq m$, the so-called hybrid approach [4]–which substitutes values into some variables before starting Gröbner basis computation–will be employed.

5.1 Assumptions for the Proposed Algorithm

The correctness and efficiency of the proposed algorithm rest on the following heuristic assumptions, which are informed by empirical evidence and are expected to hold for randomly generated MQ systems:

(H1) Degree-increasing phase (HDF4): During the degree-increasing phase, we assume that the number of new basis elements that should appear at each degree d coincides with the predicted values $\{N_d\}_{d=1}^{D}$ computed by the LGB algorithm. That is, the sequence of polynomials behaves analogously to a generic homogeneous sequence, and the Hilbert function of the ideal $\langle F \rangle$ matches that of a generic ideal of the same type. This assumption justifies the use of $\{N_d\}_{d=1}^{D}$ as a guide for limiting the number of S-polynomials to be computed.

(H2) Sufficiency of selected pairs (SelectPairLimited): For degrees $d > 4$, we assume that the number of pairs selected by SelectPairLimited, namely N_d+e, is sufficient to recover all new basis elements expected at degree d as predicted by LGB. In other words, although zero reductions may occur due to dependencies over a finite field, the parameter e ensures that the number of non-zero reductions is still at least N_d with high probability.

(H3) Degree-decreasing phase (MF4): During the degree-decreasing phase, we assume that each iteration of MF4 produces new basis elements with high probability, except for a small number of unavoidable zero reductions. The iterations are repeated until the set $\widetilde{G}$ forms a Gröbner basis of the input ideal.

These assumptions are consistent with prior theoretical studies on the behavior of generic sequences and are supported by our computational experiments. They justify the design and effectiveness of the Hilbert-driven selection in HDF4, the bounded sampling of S-polynomial pairs in SelectPairLimited, and the localized iteration in MF4.

5.2 Efficient F4 for MQ

In this section, we describe Algorithm 6, which is an overall algorithm for the efficient computation of MQ problems. Here, an MQ problem refers to the case where the number of polynomials m is greater than the number of variables n. This algorithm is designed to reduce the computational cost incurred during the generation and reduction of S-polynomials when computing the Gröbner basis G for the input sequence $F = \{f_1, f_2, \ldots, f_m\}$, where each f_i is of degree 2.

As the first step, the LGB is employed to determine the distribution $\{N_d\}_{d=1}^{D}$ of basis elements characteristic of a generic sequence. This distribution information serves as a

Algorithm 6 `Efficient_F4_for_MQ`

Require: $F = \{f_1, ..., f_m\} \subset R$ a multivariate quadratic problem $\mathrm{MQ}(n, m)$ with $n < m$ and $\deg(f_i) = d_i = 2$, a parameter $e \in \mathbb{N}$
Ensure: $A \subset R$
1: $\{N_d\}_{d=1}^{D} \leftarrow \mathtt{LGB}(n, m, \{d_i\}_{i=1}^{m})$
2: $G, \widetilde{G}, B \leftarrow \mathtt{HDF4}(F, \{N_d\}_{d=1}^{D}, e)$
3: $\widetilde{G} \leftarrow \mathtt{MF4}(G, \widetilde{G}, B, w)$
4: $A \leftarrow$ extract a reduced Gröbner basis from $\widetilde{G}$
5: **return** A

guideline to adjust the number of S-polynomials generated in subsequent steps; that is, the number of minimal Gröbner basis elements at each degree is computed as $\{N_d\}_{d=1}^{D}$.

Next, the algorithm executes the increasing-degree phase using `HDF4`. In this phase, S-polynomials are generated in the same way as in the conventional `F4`, and the reduction is carried out by constructing the matrix P and applying `RowEchelonForm`. During the execution of `HDF4`, the computed distribution $\{N_d\}_{d=1}^{D}$ is utilized to limit the number of S-polynomials to the necessary minimum, thereby avoiding unnecessary computations.

Subsequently, the algorithm performs the decreasing-degree phase via `MF4`. In `MF4`, the previously obtained basis set G and auxiliary set $\widetilde{G}$ are reused, and new S-polynomials are generated locally to efficiently advance both the generation and the reduction processes.

Finally, the polynomials that constitute the reduced Gröbner basis from $\widetilde{G}$ and returned as the output.

5.3 HDF4

The input to `HDF4` is the set of polynomials F from an MQ problem $\mathrm{MQ}(n, m)$, the numbers $\{N_d\}_{d=1}^{D}$ representing the count of basis elements computed at each degree d, and a parameter e used to adjust the number of S-polynomials. In this section, we explain `HDF4` with a focus on its differences from the F4 algorithm.

When the degree d of the S-polynomials to be computed is less than or equal to 4, the function `SelectPair` is used, and the computation proceeds in the same manner as in `F4`. For degrees greater than 4, however, the function `SelectPairLimited` is employed. Instead of computing S-polynomials for all pairs of degree d, only $N_d + e$ S-polynomials are computed, where $N_d + e$ is slightly larger than the number N_d of basis elements that can be obtained. The pairs of degree d that are not selected by `SelectPairLimited` are removed from the candidate set B. Since the number of basis elements obtainable at degree d is N_d, computing additional S-polynomials beyond that count would result in zero reductions. Moreover, because a polynomial ring over a finite field has only a limited number of coefficient elements, unlike a generic sequence over an infinite field, unexpected zero reductions may occur. Therefore, by computing $N_d + e$ S-polynomials, the impact of zero reductions can be mitigated to some extent. This issue becomes less pronounced as the number of elements in the finite field increases.

Remark 2. The reason for using `SelectPairLimited` only for degrees greater than 4 is that there is a tendency to select pairs that would lead to zero reduction when $d = 4$. Our computational experiments have shown that for $d > 4$, it is possible to select pairs that do not result in zero reductions.

Remark 3. The parameter e compensates for zero reductions caused by the finite field effect. The probability model $1/q^{1+e}$ is an empirical approximation that becomes valid

Algorithm 7 `HDF4`

Require: $F \subset R, \{N_d\}_{d=1}^{D} \subset \mathbb{N}, e \in \mathbb{N}$
Ensure: $G \subset R, \widetilde{G} \subset R, B \subset \mathbb{N}^2$
$G \leftarrow F$
$\widetilde{G} \leftarrow F$
$B \leftarrow \{(i,j) \mid 1 \leq i < j \leq \#G\}$
$d \leftarrow 2$
while $N_d \neq 0$ **do**
  if $d \leq 4$ **then**
    $B', B \leftarrow$ `SelectPair`(B, d)
  else
    $B', B \leftarrow$ `SelectPairLimited`$(B, d, N_d + e)$
  end if
  $L_S \leftarrow \{\mathrm{Spoly}(f_i, f_j) \mid (i,j) \in B'\}$
  $P, M_B \leftarrow$ `ComputeP`$(L_S, \widetilde{G})$
  $Q \leftarrow$ `RowEchelonForm`(P)
  $H \leftarrow$ polynomial form of Q
  for $h \in H$ **do**
    if $\mathrm{LM}(h) \in M_B$ **then**
      $G \leftarrow G \cup \{h\}$
      $B \leftarrow B \cup \{(i, \#G) \mid 1 \leq i < \#G\}$
    end if
    $\widetilde{G} \leftarrow \widetilde{G} \cup \{h\}$
  end for
  $d \leftarrow d + 1$
end while
return $\{G, \widetilde{G}, B\}$

only when the degree d is sufficiently large (typically $d > 4$), since in this range the generated matrices behave almost randomly with respect to their coefficients. For small degrees ($d \leq 4$), the structure of the S-polynomials is highly constrained and the independence assumption underlying this model does not hold; therefore, in that regime we use the full set of pairs without limitation. This distinction reflects an observed transition from structured to statistically random behavior of coefficient matrices, not a change in the theoretical validity of the algorithm.

5.4 SelectPairLimited

We focus on the differences from the conventional `SelectPair` and explain the pair selection algorithm `SelectPairLimited`. In `SelectPair`, all candidate pairs with degree d are sequentially selected and the computation is carried out using every such pair without restriction. In contrast, `SelectPairLimited` introduces a parameter N to impose an upper limit on the number of pairs selected, thereby avoiding unnecessary computations and improving efficiency.

Specifically, we first extract from the candidate pair set B the pairs that generate S-polynomials of degree d into an auxiliary set B''. Then, for each pair in B'', we evaluate it using the function `Criterion` and, while excluding those pairs that are predicted to yield zero reduction, we add only the useful pairs to the final selection set B'. Moreover, once the size of B' reaches the parameter N, the selection process terminates, thus limiting the

Algorithm 8 `SelectPairLimited`

Require: $B \subset \mathbb{N}^2, d \in \mathbb{N}, N \in \mathbb{N}$
Ensure: $B' \subset \mathbb{N}^2, B \subset \mathbb{N}^2$
1: $B' \leftarrow \varnothing$
2: $B'' \leftarrow \{b \in B \mid \deg(b) = d\}$ ($\deg(b)$ is calculated as (#))
3: $B \leftarrow B \setminus B''$
4: **while** $B'' \neq \varnothing$ and $\#B' \neq N$ **do**
5: $(i, j) \leftarrow$ an element in B''
6: **if** `Criterion`$((i, j), B') =$ `false` **then**
7: $B' \leftarrow B' \cup \{(i, j)\}$
8: **end if**
9: $B'' \leftarrow B'' \setminus \{(i, j)\}$
10: **end while**
11: **return** $\{B', B\}$

number of pairs considered for computation. This method serves to suppress the number of S-polynomials computed, thereby reducing the overall computational cost.

Remark 4. In line 5, although a pair is selected at random from B'', in our implementation the pairs are ordered so that those with a smaller order are chosen preferentially. Let (i, j) and (k, l) be two pairs contained in B''. Suppose the S-polynomial generated from (i, j) is of the form $t_i f_i - t_j f_j$ and that generated from (k, l) is of the form $t_k f_k - t_l f_l$. Then, we define $(i, j) < (k, l)$ if either

$$\max(t_i, t_j) < \max(t_k, t_l)$$

or, in the case where $\max(t_i, t_j) = \max(t_k, t_l)$, if

$$\min(t_i, t_j) < \min(t_k, t_l).$$

This selection method is designed to prioritize S-polynomials having smaller multiplier monomials and similar structures. As a result, S-polynomials with smaller leading terms are preferentially selected, which is expected to reduce the size of $M_A \cup M_B$ and, consequently, the size of the matrix P.

5.5 MF4

We describe `MF4`, which is specialized for the decreasing-phase computations of S-polynomials in contrast to `F4`. `MF4` leverages the basis set G and the auxiliary set $\widetilde{G}$ obtained from `HDF4`, and it performs computations restricted to the decreasing phase by locally re-selecting pairs based on the previously computed results.

Specifically, `MF4` first employs `SelectPairLimited` to restrict the number of candidate pairs in the decreasing phase to a parameter w, thereby generating only the minimally necessary set of S-polynomials. Subsequently, the generated S-polynomials are assembled into a matrix by `ComputeP` and then reduced via `RowEchelonForm`. The newly obtained basis elements are used to update both the set G_m and the auxiliary set $\widetilde{G}$; in this algorithm, G_m collects the elements obtained in each loop. Furthermore, using the function `SelectPairM`, pairs are selected in a manner that is tailored specifically for the decreasing-phase computations, and the corresponding S-polynomials are generated.

Remark 5. We explain how the parameter w is determined. Among the matrices computed in each loop of `MF4`, the one with the longest computation time is typically the first

Algorithm 9 `MF4`

Require: $G \subset R, \widetilde{G} \subset R, B \subset \mathbb{N}^2, w \in \mathbb{N}$
Ensure: $\widetilde{G} \subset R$
1: $B', B \leftarrow$ `SelectPairLimited`(B, d, w)
2: $L_S \leftarrow \{\mathrm{Spoly}(f_i, f_j) \mid (i,j) \in B'\}$
3: **while** true **do**
4: $P, M_B \leftarrow$ `ComputeP`$(L_S, \widetilde{G})$
5: $Q \leftarrow$ `RowEchelonForm`(P)
6: $H \leftarrow$ polynomial form of Q
7: $G_m \leftarrow \varnothing$
8: **for** $h \in H$ **do**
9: **if** $\mathrm{LM}(h) \in M_B$ **then**
10: $G_m \leftarrow G_m \cup \{h\}$
11: **end if**
12: $\widetilde{G} \leftarrow \widetilde{G} \cup \{h\}$
13: **end for**
14: **if** $G_m = \varnothing$ **then**
15: **break**
16: **end if**
17: $L_S \leftarrow$ `SelectPairM`$(G_m, \widetilde{G})$
18: **end while**
19: **return** $\{\widetilde{G}\}$

matrix. Therefore, if w is set too large, the computation time for the first matrix becomes excessively long. On the other hand, if w is set too small, the number of loops increases, and as a result, the number of matrix constructions increases. In other words, the optimal value of w depends on the implementation. In our implementation, letting D be the largest degree such that $N_D \neq 0$, we chose $w = N_D/2$.

5.6 SelectPairM

We describe the pair selection strategy called `SelectPairM`, which differs from `SelectPair` approach. `SelectPairM` utilizes the polynomial set G_m and the auxiliary set $\widetilde{G}$. For each $f \in G_m$, it verifies whether there exists an $\widetilde{f} \in \widetilde{G}$ such that $\mathrm{LM}(x_n \cdot f) = \mathrm{LM}(\widetilde{f})$. If this condition is satisfied, an S-polynomial is generated in the form $x_n \cdot f - \widetilde{f}$. Although this S-polynomial selection method may seem special, it is merely a normal selection strategy with restrictions. We assume that x_n is the smallest variable with respect to the monomial ordering used in the polynomial ring R.

This selection method offers two main advantages. First, by restricting the multiplier monomial to x_n, the generated S-polynomials have smaller leading terms. Second, by limiting the multiplier monomial exclusively to x_n, a set of similar S-polynomials is generated, which is expected to reduce the overall size of the matrix P. In contrast to conventional methods that indiscriminately extract candidate pairs from the entire set, this approach is anticipated to yield a smaller $M_A \cup M_B$ and, consequently, a smaller matrix P. It should be noted that this way of generating S-polynomials differs from that used in the MXL algorithms.

Algorithm 10 `SelectPairM`

Require: $G_m \subset R, \widetilde{G} \subset R$
Ensure: $L_S \subset R$
1: $L_S \leftarrow \varnothing$
2: **for** $f \in G_m$ **do**
3: **if** there exists $\widetilde{f} \in \widetilde{G}$ such that $\mathrm{LM}(x_n \cdot f) = \mathrm{LM}(\widetilde{f})$ **then**
4: $L_S \leftarrow L_S \cup \{x_n \cdot f - \widetilde{f}\}$
5: **end if**
6: **end for**
7: **return** $\{L_S\}$

6 Implementation

This section describes the implementation of the proposed algorithm. In our experiments, more than 90% of the total running time is spent in matrix operations, and therefore we focus in particular on their implementation.

6.1 Matrix Operations

As described in Section 3.6, both T_{CB} and T_{BD} are essentially matrix multiplications of the form

$$D \leftarrow D - CB', \qquad B'' \leftarrow B'_2 - B'_1 D''.$$

These matrices are in general not square but rectangular, and their density is high, typically 30%-95%. Furthermore, as shown in Appendix A, in our implementation T_{AB} is also realized as a matrix-multiplication–type operation.

All these matrix multiplications are implemented using a naive triple-loop algorithm combined with blocking to improve cache locality and SIMD efficiency. The block size is chosen as a square block of 1 byte $\times$ 32 = 32 bytes, so as to match the 256-bit width of AVX2 registers. During matrix construction, if a block consists entirely of zeros, the block is not materialized. This reduces memory consumption and avoids useless computations on all-zero blocks.

In contrast, T_R is implemented as a simple Gaussian elimination (elementary row operations) on blocked matrices. These matrices are in general rectangular. In our experiments, they are typically wide rectangular matrices, and their density is close to 100%.

We do not use Strassen-type matrix multiplication algorithms [21]. We implemented and evaluated such algorithms, but for the rectangular and moderately sized matrices arising in our computations, they did not provide any noticeable speedup compared with the naive blocked multiplication.

6.2 Data Representation

Each field element is represented as an 8-bit signed integer in the range $[0, 30]$. Matrices are stored in row-major order in memory. To efficiently use the AVX2 instruction `vpmaddubsw` (`_mm256_maddubs_epi16`), the right-hand matrix is preprocessed by interleaving two consecutive rows. More precisely, the $2i$-th and $(2i+1)$-st rows are rearranged into a 256-bit register, and placed alternately. With this layout, two products can be computed simultaneously by a single instruction.

Table 1: Main AVX2 instructions used in the matrix multiplication kernel.

Instruction	Role
`_mm256_unpacklo_epi8`, `_mm256_unpackhi_epi8`	Interleave two rows of the right-hand matrix
`_mm256_broadcastw_epi16`	Broadcast coefficients of the left-hand matrix
`_mm256_maddubs_epi16`	Byte-wise multiply–accumulate into 16-bit lanes
`_mm256_add_epi16`	Accumulate partial sums
`_mm256_srli_epi16`, `_mm256_and_si256`	Branchless reduction modulo 31
`_mm256_packs_epi16`	Pack 16-bit values into 8-bit values
`_mm256_cmpgt_epi8`, `_mm256_sub_epi8`	Final normalization to $[0, 30]$

6.3 SIMD-Based Matrix Multiplication

The dominant computational cost of our algorithm lies in dense matrix multiplications over $\mathbb{F}_{31}$. To accelerate these operations, we implement a SIMD-based kernel using AVX2 instructions.

Our kernel computes two output rows simultaneously. To enable this, two consecutive rows of the right-hand matrix are first interleaved by AVX2 unpack instructions, so that elements from both rows are arranged in an alternating layout. This preprocessing allows us to process both rows in parallel within a single SIMD register.

For the left-hand matrix, coefficients are broadcast across SIMD lanes using AVX2 broadcast instructions together with shift and permutation operations. This enables reuse of each coefficient across multiple multiply–accumulate operations.

The core arithmetic operation is performed by `vpmaddubsw` (`_mm256_maddubs_epi16`), which computes products of 8-bit integers and accumulates them into 16-bit lanes. Partial sums are then accumulated in 16-bit lanes using AVX2 integer addition. This design allows us to process 32 columns in parallel within a single 256-bit register.

Overall, this SIMD kernel efficiently combines data layout transformation, broadcasting, and vectorized multiply–accumulate operations to maximize data throughput on AVX2-capable architectures.

6.4 Modular Arithmetic Optimization

To efficiently perform arithmetic over $\mathbb{F}_{31}$, we use a branchless modular reduction scheme adapted to SIMD instructions.

Let c be a 16-bit intermediate value. We exploit the identity

$$c \bmod 31 = (c \,\&\, 31) + (c \gg 5),$$

which follows from $31 = 2^5 - 1$. This reduction is applied iteratively until the result lies within the range $[0, 30]$.

In the implementation, the reduction is realized by SIMD shift, mask, and addition operations on 16-bit lanes, followed by packing the results back into 8-bit values. Finally, a branchless normalization step subtracts 31 from entries greater than 30, ensuring that all values lie in the canonical range $[0, 30]$.

This approach enables fast modular arithmetic fully compatible with the vectorized matrix multiplication kernel described above.

Table 2 shows a compact profiling summary of our implementation on a representative large instance. The cost of matrix construction is included in the corresponding transformation steps, since these operations are tightly integrated in our implementation.

The results show that matrix-related operations account for more than 90% of the total running time. In particular, the dominant costs come from the dense blocked matrix multiplications corresponding to T_{CB} and T_{BD}, together with the row echelon computation T_R. This confirms that dense linear algebra dominates the overall complexity in the parameter regime of interest, justifying our focus on SIMD optimization.

Table 2: Compact profiling summary of our implementation on a large instance MQ(21, 24) over $\mathbb{F}_{31}$ using 16 threads. The cost of matrix construction is included in the corresponding transformation steps.

Component	Time ratio (%)
T_{AB}	1.8
T_{CB}	41.6
T_R	17.5
T_{BD}	29.7
Other	9.4

For smaller instances, the proportion of matrix-related operations is around 70%, but it increases with the problem size. This trend indicates that the dominance of matrix operations becomes more pronounced as the problem size grows.

6.5 Parallelism

To exploit multi-core CPUs, we parallelize the blocked matrix multiplication using OpenMP. The outermost loop iterates over the output matrix in blocks along the row dimension. We assign these row blocks to threads using `#pragma omp parallel for`. Each thread is thus responsible for updating a disjoint set of row blocks, which avoids write-write conflicts without requiring synchronization.

7 Comparison of the Computation Times

In order to evaluate the efficiency of our proposed algorithm, we conducted a series of computational experiments comparing our implementation with two publicly available programs, namely M4GB and Magma's F4. All experiments were executed on a server equipped with an AMD EPYC 7763 CPU and 2 TB of RAM, running Ubuntu 20.04 LTS. Our implementation was written in C++ and compiled with GCC 13.3.0 using the flags `-O3 -mavx2 -fopenmp`. Unless otherwise stated, all Gröbner basis computations used the degree reverse lexicographic order with $x_1 > \cdots > x_n$. Parallelization was implemented using OpenMP. In the single-core experiments, we fixed the number of threads to 1, while in the parallel experiments we used 12 threads. For M4GB, we used the publicly available implementation [15]. For Magma, we used version 2.27-9. In all comparisons, we used the same monomial order and matched the thread configuration as closely as supported by each system. The test problem consisted of a randomly generated MQ problem over the finite field $\mathbb{F}_{31}$, in which the number of equations was set to be one more than the number of variables, i.e., $m = n + 1$.

The evaluation focused on both computation time and memory usage. When the programs were run on a single core, the computation time and memory usage was recorded and is illustrated in Figure 3. Furthermore, to assess the impact of parallelization, the computation time and memory consumption were measured when the programs were executed using 12 cores, with these results presented in Figure 4.

Based on the single-core comparison results, it has been verified that the algorithm proposed in this paper outperforms existing Gröbner basis computation algorithms in terms of speed. In particular, as the number of variables increases–that is, as the MQ problems become more challenging–the performance improvement becomes even more significant. Moreover, in terms of memory usage, our proposed algorithm requires less memory since the matrices generated during the computation are relatively small. Similar trends were observed in the parallel computation experiments as well.

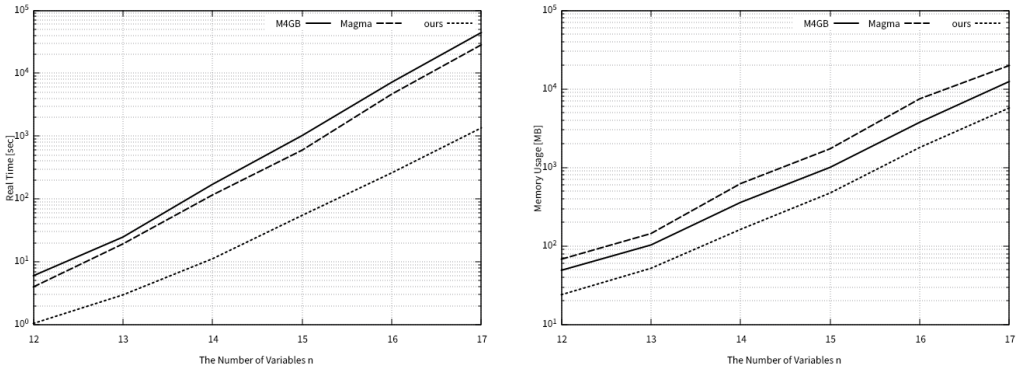

Figure 3: Comparison of computation time (left) and memory usage (right) on a single core.

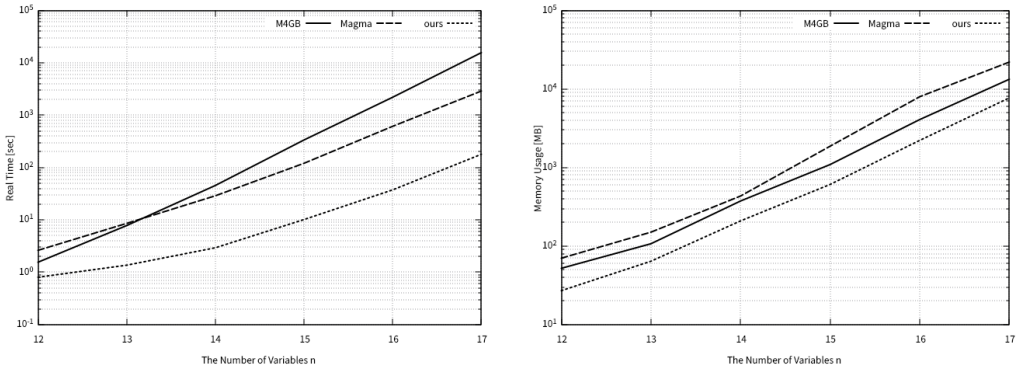

Figure 4: Comparison of computation time (left) and memory usage (right) on parallel 12-core.

Table 3 shows a comparison of the matrix (P) sizes generated by the `F4` and `HDF4` when solving an MQ problem over the field $\mathbb{F}_{31}$ with 12 variables and 13 equations. For `HDF4`, the redundancy parameter was set to $e = 0$. The columns labeled row, col, M_A, M_B, and L_S represent, respectively, the number of rows in the matrix P (i.e., $\#M_A + \#L_S$), the number of columns (i.e., $\#M_A + \#M_B$), the number of reducible monomials M_A, the number of non-reducible monomials M_B, and the number of computed S-polynomials L_S.

The results confirm that `HDF4` computes significantly fewer S-polynomials than the `F4`, as indicated by the reduced size of L_S. As shown in the table, `HDF4` has a much smaller $\#M_A$. This is because the reduction in the number of S-polynomials decreases the variety of monomials to be reduced, which in turn reduces the number of polynomials used for the reduction. It should be noted that, for each degree d, `HDF4` constructs smaller matrices than `F4` while yielding the same results. Consequently, the overall matrix P becomes smaller, leading to the performance improvement shown in Figures 3 and 4.

Similarly, `MF4` is expected to show a comparable reduction in matrix size, since the S-polynomials it generates are restricted to those multiplied by the smallest variable in the term order, x_n. This restriction decreases the variety of monomials to be reduced and the number of polynomials used in the reduction. However, during the decreasing phase—when the degree of S-polynomials becomes smaller—the correspondence between matrices of different sizes and their resulting reductions does not hold as in the increasing phase. Therefore, a direct comparison of matrix sizes is not straightforward in this case.

Table 4 reports the number of non-zero entries and the density of each matrix A, B, C, and D in the matrix P generated by `F4` and `HDF4` when solving MQ$(12, 13)$. `HDF4`

Table 3: Comparison of matrix sizes generated by `F4` and `HDF4` when solving an MQ(12, 13) problem over $\mathbb{F}_{31}$.

	F4					HDF4				
d	row	col	M_A	M_B	L_S	row	col	M_A	M_B	L_S
5	2947	3786	2343	1443	604	1220	2507	1064	1443	156
6	6361	6881	4879	2002	1482	1886	3602	1600	2002	286
7	10346	9488	7343	2145	3003	1992	3708	1563	2145	429

Table 4: The number of non-zero entries and density (%) of matrix A, B, C, D generated by `F4` and `HDF4` when solving an MQ(12, 13) problem.

	F4				HDF4			
d	A	B	C	D	A	B	C	D
5	46,473 (0.96%)	1,070,412 (33.8%)	386,595 (29.1%)	288,754 (33.1%)	6,617 (0.6%)	590,031 (38.4%)	72,651 (43.7%)	122,372 (54.3%)
6	226,344 (1.0%)	3,847,191 (41.0%)	1,789,696 (25.7%)	1,192,569 (40.1%)	33,619 (1.3%)	1,581,000 (49.3%)	219,024 (47.8%)	392,813 (68.6%)
7	521,807 (1.1%)	7,780,400 (52.3%)	5,087,791 (24.4%)	2,957,164 (45.9%)	57,832 (2.3%)	2,268,365 (67.6%)	426,983 (63.6%)	733,597 (79.7%)

produces matrices with fewer non-zero entries than `F4`, showing that the same Gröbner basis is computed using a smaller amount of information. On the other hand, when looking at the density of each block, we observe that highly dense matrices are often generated. In particular, for `HDF4`, the density of block A appears to be very small in the table (0.6%, 1.3%, and 2.3%). However, as explained in Section 6, our implementation constructs matrices in a block-wise manner and does not generate blocks that consist entirely of zeros. Therefore, if the density of A is recomputed based on the number of elements and the number of generated blocks, the effective densities become 79.7%, 65.2%, and 84.6%, respectively, showing that the block matrix is in fact highly dense. Finally, the density of block D reported in the table corresponds to the value before applying T_{CB}. Our experiments confirm that after applying T_{CB}, D becomes almost 100% dense.

To quantify the robustness of the Hilbert-driven limiting over different inputs, we conducted additional experiments on multiple random MQ instances with identical parameters. For each parameter set (n, m) with $n = 12, \ldots, 17$ and $m = n + 1$, we generated 100 independent instances and ran HDF4 with the same configuration as in the main experiments, using a fixed redundancy parameter $e = 5$. In all cases, the algorithm completed successfully without failure, yielding a success rate of 100% for all tested parameters. This indicates that the Hilbert-driven limiting is stable across different inputs with the same parameters.

8 New Record in the Fukuoka MQ Challenge

The Fukuoka MQ Challenge, which has been held since 2015, serves as a benchmark for assessing the computational hardness of MQ problems. In this competition, problem instances are categorized into Types I–VI based on the underlying finite field $\mathbb{F}_q$, the number of variables n, and the number of equations m. In our work, we focused on a Type VI instance defined over the finite field $\mathbb{F}_{31}$ with a variable-to-equation ratio approximately given by $n \sim 1.5m$. The previous record for Type VI was set with $n = 30$ variables and $m = 20$ equations, where the reported solution time was roughly 11 days using M4GB on a system equipped with dual Intel Xeon E5-2650 v3 CPUs and 256 GB of RAM (they guessed 12 variables and the number of trials for the hybrid approach was

not disclosed).

In our experiments, we targeted a Type VI instance with $n = 31$ and $m = 21$. Our proposed algorithm was implemented in C++ and executed on a system powered by an AMD EPYC 7763 CPU with 2 TB of RAM. To guarantee a high success rate, we computed a slightly larger number of S-polynomials by setting the parameter $e = 5$. Our implementation, which is optimized for dense matrices over $\mathbb{F}_{31}$ using AVX2, achieved substantial acceleration through parallel processing in both matrix multiplication and the computation of the row echelon form. Moreover, by employing a hybrid approach (guessing 11 variables and solving MQ$(20, 21)$), we managed the memory constraints of our experimental platform by running four simultaneous Gröbner basis computations. As a result, we succeeded in solving the seed 0 instance of the Type VI MQ Challenge in approximately 9 hours, thereby breaking the previous record.

Furthermore, applying the same techniques, we tackled another Type VI problem with 33 variables and 22 equations, and we were able to solve the seed 0 instance in about 2 days and 3 hours by employing a hybrid approach (guessing 12 variables and solving MQ$(21, 22)$). These achievements not only demonstrate the superior efficiency of our algorithm but also highlight its robustness in handling increasingly challenging MQ problems.

Extended Experiments. We further extended our experiments to larger Type VI instances, which were executed on a high-performance computing system equipped with four AMD EPYC 7763. In particular, we solved the problem with $n = 34$ variables and $m = 23$ equations in approximately 21 days and 12 hours by substituting 13 variables and solving MQ$(21, 23)$. Additionally, the instance with $n = 36$ and $m = 24$ was solved in about 11 days and 12 hours by substituting 15 variables and solving MQ$(21, 24)$. These results further demonstrate the scalability and robustness of our proposed algorithm when tackling increasingly larger MQ systems.

According to the computational complexity estimation formula given in [24], the problem with $n = 36$ and $m = 24$ (guessed 15 variables) has an estimated computational complexity approximately 47,627 times greater than the problem with $n = 30$ and $m = 20$ (guessed 12 variables). This estimation highlights the rapidly increasing difficulty of larger MQ instances and underscores the computational significance of the results achieved in our experiments.

9 Conclusion

In this paper, we proposed an enhanced variant of the F4 algorithm optimized for solving multivariate quadratic (MQ) problems, which are fundamental to the security of various multivariate cryptographic schemes. By integrating a Hilbert-driven strategy and a refined pair selection mechanism, our method significantly reduces the number of zero reductions and the size of intermediate matrices, thereby improving the overall efficiency of Gröbner basis computations.

Our algorithm comprises two main components: HDF4, which restricts the number of S-polynomials in the degree-increasing phase based on Hilbert function predictions, and MF4, which efficiently completes the computation in the degree-decreasing phase using a localized strategy. These components are supported by heuristic assumptions justified by the genericity of the input and validated through experimental observations.

Through extensive benchmarks, we demonstrated that our algorithm outperforms state-of-the-art systems such as M4GB and Magma's F4 in both single-core and multi-core environments. In particular, our results show that the proposed algorithm is especially effective in the regime $m/n \approx 1$, which is the most relevant setting for algebraic attacks after guessing. In this regime, we achieved new records in the Fukuoka MQ Challenge for Type VI instances over $\mathbb{F}_{31}$, highlighting the practical impact of our approach.

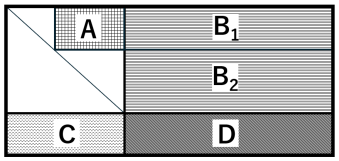

Figure 5: Typical shape of a matrix P in HDF4 and MF4

Although we do not provide extensive experiments in the overdefined regime $m/n \approx a > 1$, the proposed Hilbert-driven strategy is expected to remain effective because it limits unnecessary reductions. However, compared to approaches such as Wiedemann-XL, our method relies on dense linear algebra and may require significantly larger memory. A detailed comparison in this regime is left for future work.

Future work includes further optimization of the matrix kernels on newer SIMD-capable processors. In particular, although our current experimental platform did not support AVX-512, an AVX-512-based implementation may yield additional speedups on suitable hardware. Moreover, on processors supporting AVX-VNNI, it would be interesting to redesign the current byte-level multiply-accumulate kernel using instructions such as `_mm256_dpbusd_epi32`, which may reduce the instruction count compared with the current AVX2-based implementation. For computations over $\mathbb{F}_{2^8}$, GFNI instructions such as `_mm256_gf2p8mul_epi8` also provide a promising direction for accelerating field multiplication, especially when the field representation is chosen compatibly.

The Magma implementation of the proposed algorithms is publicly available at:
https://github.com/sakataUT/hdf4_mf4_m

A Efficient Implementation of the Matrix Operation T_{AB}

Generally, T_{AB} proceeds in two steps: first the inverse matrix A^{-1} is produced, and next the lower block is updated by the assignment $B \leftarrow A^{-1}B$.

Our computational experiments show that every matrix generated by HDF4 and MF4 falls into one of two special categories. Either A is already the identity matrix, or it is a sparse upper-triangular matrix whose shape is depicted in Figure 5. When A equals the identity, T_{AB} becomes trivial. When A exhibits the shape in Figure 5, the update can be realised with a single row operation $B_1 \leftarrow B_1 - A\,B_2$, provided that B is partitioned conformally as $\begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$. In either situation, A^{-1} is unnecessary, so the entire reduction can be executed using only a matrix–matrix product followed by a short Gaussian elimination.

The absence of any explicit inversion routine simplifies the implementation and yields a favourable impact on both running time and memory consumption in all benchmarks we performed.

Acknowledgement

This work was supported by JST K Program Grant Number JPMJKP24U2, Japan. This work was also supported by JSPS KAKENHI Grant Number JP25K00140.

References

[1] Albrecht M. R., Cid C., Faugère J.-C., Perret L., “On the relation between the MXL family of algorithms and Gröbner basis algorithms”, J. Symb. Comput. 47(8): 926–941,

2012.

[2] Bardet M., Faugére J.-C., Salvy B., "On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations", In Proceedings of the International Conference on Polynomial System Solving, 71–74, 2004.

[3] Bardet M., Faugére J.-C., Salvy B., Yang B-Y., "Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems", In Proceedings of MEGA, 8th International Symposium on Effective Method in Algebraic Geometry, 2006.

[4] Bettale L., Faugère J.C., Perret L., "Hybrid approach for solving multivariate systems over finite fields", Journal of Mathematical Cryptology, 3, issue 3, 177–197, 2009.

[5] Beullens W., Chen M.-S., Ding J., Gong B., Kannwischer M.J., Patarin J., Peng B.-Y., Schmidt D., Shih C.-J., Tao C., Yang B.-T.: "UOV: Unbalanced Oil and Vinegar", https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/UOV-spec-web.pdf, 2023.

[6] Buchberger B., "Ein Algorithmus zum Auffinden der basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal", PhD thesis, Universität Innsbruck, 1965.

[7] Buchberger B., "Criterion for detecting unnecessary reductions in the construction of Gröbner basis", EUROSAM 79, Lecture Notes in Computer Science, 72, Springer, 3–21, 1979.

[8] Cox D., Little J., O'Shea D., "Ideals, Varieties, and Algorithms", Springer-Verlag, New York, 1997.

[9] Faugère J.-C., "A new efficient algorithm for computing Gröbner Bases (F4)", Journal of Pure and Applied Algebra, 139, 6–88, 1999.

[10] Faugère J.-C., "A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)", In Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, 75–83, 2002.

[11] Faugère J.-C., Lachartre S., "Parallel Gaussian elimination for Gröbner bases computations in finite fields", PASCO 2010: 89-97, 2010.

[12] Fröberg R., "An inequality for Hilbert series of graded algebras", Math. Scand. 56, 2, 117–144, 1985.

[13] Gebauer R., Möller H. M., "On an installation of Buchberger's algorithm", Journal of Symbolic Computation, 6, 275–286, 1988.

[14] Ito T., Shinohara N., Uchiyama S., "An efficient F4-style based algorithm to solve MQ problems", in: Proc. of 14th International Workshop on Security 2019, 37–52, 2019.

[15] Makarim R. H., Stevens M. , "M4GB: An efficient Gröbner-basis algorithm", In Proceedings of the 2017 International Symposium on Symbolic and Algebraic Computation, 293–300, 2017.

[16] Mohamed M.S.E., Cabarcas D., Ding J., Buchmann J., Bulygin S., "MXL3: An Efficient Algorithm for Computing Gröbner Bases of Zero-Dimensional Ideals", ICISC 2009: 87-100, 2009.

[17] Moreno-Socías G., "Autour de la fonction de Hilbert-Samuel (escaliers d'idéaux polynomiaux)", Thèse, École Polytechnique, 1991.

[18] Pardue K., "Generic sequences of polynomials", Journal of Algebra, 324(4), 579–590, 2010.

[19] Peter W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", SIAM Jounal on Computing, 26, issue 5, 1484–1509, 1997.

[20] Sakata K., Takagi T., "An Algorithm for the Leading Monomials of a Minimal Gröbner Basis of Generic Sequences", https://arxiv.org/abs/2505.10246, 2025.

[21] Strassen V., "Gaussian Elimination is not Optimal", Numer. Math. 13, p. 354-356, 1969.

[22] Traverso C., "Hilbert functions and the buchberger algorithm", Journal of Symbolic Computation, 22(4), 355–376, 1996.

[23] Williams V. V., Xu Y., Xu Z., Zhou R. "New Bounds for Matrix Multiplication: from Alpha to Omega", CoRR abs/2307.07970, 2023.

[24] Yasuda T., Dahan X., Huang Y-J., Takagi T., Sakurai K., "MQ Challenge: Hardness Evaluation of Solving Multivariate Quadratic Problems", NIST Workshop on Cybersecurity in a Post-Quantum World, 2015. https://www.mqchallenge.org/

[25] Yang B.-Y., Chen J.-M., "All in the XL family: theory and practice", ICISC 2004: 67–86, 2004.

[26] Yang B.-Y., Chen J.-M., Courtois N.T., "On asymptotic security estimates in XL and Gröbner bases-related algebraic cryptanalysis", ICICS 2004: 401–413, 2004.

[27] Yang B.-Y., Chen J.-M., "Theoretical analysis of XL over small fields. ACISP 2004", ACISP 2004, 277–288, 2004.

[28] Yeh J.Y-C, Cheng C.-M., Yang B.-Y., "Operating degrees for XL vs. F4/F5 for generic MQ with number of equations linear in that of variables", Number Theory and Cryptography 2013, 19–33, 2013.

[29] Yang B.-Y., Chen C.-H.O. , Bernstein B.J., Chen J.-M.: "Analysis of QUAD", FSE 2007: 290-308, 2007.