Journal of Symbolic Computation 103 (2021) 66–94

Contents lists available at ScienceDirect

Journal of Symbolic Computation

www.elsevier.com/locate/jsc

On the efficiency of solving Boolean polynomial systems with the characteristic set method ☆

Zhenyu Huang, Yao Sun (c), Dongdai Lin

SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

ARTICLE INFO

Article history:
Received 10 July 2019
Accepted 30 October 2019
Available online 8 November 2019

Keywords:
Boolean polynomial system
Characteristic set method
Computation complexity
Zero decomposition
Cryptanalysis

ABSTRACT

An improved characteristic set algorithm for solving Boolean polynomial systems is proposed. This algorithm is based on the idea of converting all the polynomials into monic ones by zero decomposition, and using additions to obtain pseudo-remainders. Three important techniques are applied in the algorithm. The first one is eliminating variables by new generated linear polynomials. The second one is optimizing the strategy of choosing polynomial for zero decomposition. The third one is to compute add-remainders to eliminate the leading variable of new generated monic polynomials. By analyzing the depth of the zero decomposition tree, we present some complexity bounds of this algorithm, which are lower than the complexity bounds of previous characteristic set algorithms. Extensive experimental results show that this new algorithm is more efficient than previous characteristic set algorithms for solving Boolean polynomial systems.

1. Introduction

Solving Boolean polynomial systems, which is solving polynomial systems in the ring $\mathbf{R}_2 = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n\rangle$, plays a fundamental role in many important fields such as coding theory, cryptology, and analysis of computer hardware. To find efficient algorithms for solving such systems is a central issue both in mathematics and in computer science. In the past 20

☆ This work was in part supported by the National Natural Science Foundation of China under Grants No. 61977060 and No. 61877058.

E-mail address: sunyao@iie.ac.cn (Y. Sun (c)).

https://doi.org/10.1016/j.jsc.2019.11.001
0747-7171/

years, efficient algorithms for solving Boolean polynomial systems have been developed, such as the Gröbner basis algorithm (Faugère, 1999, 2002; Gerdt and Zinin, 2008; Sun et al., 2016), the XL algorithm (equivalent to a sub-optimal version of F4) (Courtois et al., 2000; Courtois, 2002), the algorithms based on SAT solvers or MILP solvers (Bard et al., 2007; Soos et al., 2009; Borghoff et al., 2009), the fast exhaustive search algorithm (Bouillaguet et al., 2010), and the hybrid algorithms by combining exhaustive search and Macaulay matrix computation (Bettale et al., 2009; Joux and Vanessa, 2017). Some of these algorithms had good performance on solving some polynomial systems generated from cryptanalysis problem (Faugère and Joux, 2003; Simonetti et al., 2008; Eibach et al., 2008; Eibach and Völkel, 2010). The asymptotic complexity of solving Boolean polynomial systems, especially the complexity of solving Boolean quadratic polynomial systems, was well studied. The complexity of Gröbner basis algorithms for solving Boolean polynomial systems was investigated in (Bardet et al., 2003, 2004) by estimating the degree of regularity of the overdetermined polynomial system $\{\mathbf{P}, \mathbf{H}\}$, where $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$ is the input Boolean polynomial system and $\mathbf{H} = \langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n\rangle$ is the field equations. They presented several complexity bounds of the Gröbner basis algorithm for semi-regular quadratic systems when m/n tends to different values. Their results showed that for semi-regular quadratic systems with n variables and m polynomials, when $m \sim Nn(N > 1/4)$, the complexity of the Gröbner basis computation is single exponential, when $n << m << n^2$, the complexity is sub-exponential, and when $m \sim Nn^2$, with N a constant, the complexity is polynomial. Furthermore, in (Bardet et al., 2013), Bardet et al. presented a new algorithm to solving quadratic Boolean polynomial system based on the idea of combining exhaustive search and sparse linear algebra. Under precise algebraic assumptions which are satisfying with probability very close to 1, the deterministic variant of their algorithm has complexity bounded by $O(2^{0.841n})$ when m = n, while a probabilistic variant of their algorithm (Las Vegas type) has expected complexity $O(2^{0.792n})$, where m is the number of polynomials and n is the number of variables. Moreover, for input systems without side conditions, the best complexity of solving quadratic Boolean polynomial systems is reached by a fast exhaustive search algorithm in $4\log_2 n2^n$ bit operations (Bouillaguet et al., 2010).

The **characteristic set (CS)** method is a tool for studying polynomial, algebraic differential, and algebraic difference equation systems (Aubry et al., 1999; Boulier et al., 1995; Bouziane et al., 2001; Chou, 1988; Chou and Gao, 1990; Dahan et al., 2005; Gao et al., 2009; Hubert, 2000; Kalkbrener, 1993; Kapur and Wan, 1990; Lazard, 1991; Lin and Liu, 1993; Li et al., 2010; Möller, 1993; Wang, 1993; Wu, 1986). The idea of the method is reducing equation systems in general form to equation systems in the form of triangular sets. Then, the zero set of an equation system can be decomposed into the union of the zero sets of triangular sets. With this method, solving an equation system can be reduced to solving univariate equations in cascaded form. The CS method can also be used to compute the dimension, the degree, and the order for an equation system, to solve the radical ideal membership problem, and to prove theorems from elementary and differential geometries. In most existing work on CS methods, the common zeros of the equations are taken in an algebraically closed field which is infinite. In (Chai et al., 2008; Gao and Huang, 2012; Huang, 2012), the CS method is extended to solving the polynomial systems in finite fields. Based on the CS algorithm proposed in (Gao and Huang, 2012), an algorithm for solving Boolean polynomial systems with noise was proposed and well studied (Huang and Lin, 2013, 2017). In (Zhao et al., 2018), the parallel version of the CS algorithm proposed in (Gao and Huang, 2012) was presented and efficiently implemented under the high-performance computing environment. Research about the complexity of the CS method is rare, and most of the existing results are about the complexity of computing one triangular set (Gallo and Mishra, 1991; Szanto, 1999). The unique result about the complexity of the whole zero decomposition process is that for an input Boolean polynomial system with n variables and m polynomials, the bit-size complexity of the top-down characteristic set ($\mathbf{TDCS_2}$) algorithm is bounded by $O(2^{log_2(m)n})$ (Gao and Huang, 2012). Since for most problems, m is big hence this bound is much worse than the complexity bounds of the methods introduced in last paragraph.

For the CS method, the bottleneck problem that limits its practical efficiency and causes its high asymptotic complexity is the intermediate expression swell, which is caused from computing pseudo-remainders by polynomial multiplications. This problem is effectively avoided in the CS algorithm **MFCS** (Multiplication-free CS), which is an algorithm for solving Boolean polynomial systems, proposed in (Gao and Huang, 2012). Hence the practical efficiency of **MFCS** is much better than $\mathbf{TDCS_2}$.

The mainly idea of **MFCS** is to use the properties of $\mathbb{F}_2$ to convert the initials of all Boolean polynomials into constant 1 by zero decomposition, then compute pseudo-remainders by addition. For this algorithm, the principle factor which effects its efficiency becomes the number of branches, that is the number of polynomial sets generated in the zero decomposition process. However, in (Gao and Huang, 2012), the authors only introduced the basic algorithm **MFCS**. Some important techniques, which were used in the implementation of **MFCS** and can greatly reduce the number of branches hence improve the efficiency of the algorithm, were not presented. Moreover, the complexity of **MFCS** is unknown, thus how these techniques influence the complexity of the algorithm is also unknown. Therefore, the motivation of this paper is to estimate the complexity of the CS algorithms based on the multiplication-free idea, analyze the variation of the complexity after applying these techniques, improve these techniques, and finally propose an algorithm which has better theoretical and practical complexity than the previous CS algorithms.

The main contributions of this paper are as follows:

1) We propose an algorithm **BCS** by modifying **MFCS** with three major techniques. The first one is adding a simplification process into algorithm, in which we use the linear polynomials generated in the zero decomposition process to eliminate variables. The second one is using an alternative `choose` function to determine the order of choosing polynomial for zero decomposition. The third one is that in the zero decomposition process, instead of adding a polynomial $I+1$ into the current branch and I into the new generated branch, we add the add-remainder of $I+1$ w.r.t. a monic triangular set **M** into the current branch and the add-remainder of I w.r.t. **M** into the new generated branch, where an add-remainder is the output of a new kind of elimination operation.
2) We define a binary tree called the zero decomposition tree, and convert the problem of estimating the complexity of **BCS** into two problems. The first one is estimating the complexity of solving one branch of the tree, and the second one is estimating the depth of the tree. For the first problem, we prove that for **BCS** the complexity of solving one branch is bounded by $O(dm(m+b)\log(n)n^{d+3})$, where d is the degree of the system, m is the number of polynomials, n is the number of variables, and b is the depth of the zero decomposition tree. For the second problem, we show that for any `choose`, b is bounded by $m\sum_{j=1}^{d-1}\binom{n}{j}$. Moreover, we introduce a vector index based on the lowest degree of the non-monic polynomials and monic polynomials, then by analyzing the variation of the sum of the entries in this vector index after zero decomposition and other operations, we show that b is bounded by $(d-1)n$, if `choose` always choose the polynomial with lowest degree, or the polynomial whose initial has lowest degree. Then for any input system, the complexity of **BCS** is bounded by $O(dm(m+d-1)\log(n)n^{d+3}2^{(d-1)n})$. Especially, for quadratic polynomial systems, the complexity of **BCS** is bounded by $O(n(m+n)2^n)$.
3) We implemented **BCS** with C language and tested its efficiency by solving several groups of Boolean polynomial systems generated from cryptanalysis and reasoning problems, and some random generated Boolean polynomial systems. We compared the timings of **BCS** and other algorithms, such as the **MFCS** algorithm used in the experiments of (Gao and Huang, 2012), the Boolean Gröbner basis routine in Magma V2.20, the Boolean Gröbner basis routine implemented by the Polybori library in SAGE V8.7, and a SAT-solver Cryptominisat V5.6.8. From the experimental results, we can observe that **BCS** is the most efficient algorithm for solving these problems generated from cryptanalysis and reasoning, and is compare with other algorithms for solving these random generated problems.

The rest of this paper is organized as follows. In Section 2, we introduce the problem of solving Boolean polynomial systems. In Section 3, the **BCS** algorithm is proposed. In Section 4, we prove the correctness of **BCS**. In Section 5, we present some complexity bounds of **BCS**. In Section 6, we present the experimental results. In Section 7, conclusions are given.

2. The problem of solving Boolean polynomial systems

Let $\mathbb{F}_2$ be the finite field of two elements $\{0,1\}$, and $\{x_1,x_2,\ldots,x_n\}$ be a variable set. Unless otherwise stated, these variables are ordered as $x_1<x_2<\cdots<x_n$. Consider the Boolean polynomial ring

$\mathbf{R}_2 = \mathbb{F}_2[x_1, x_2, \ldots, x_n]/\langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n\rangle$. An element in $\mathbf{R}_2$ is called a Boolean polynomial. In this paper, we consider the following problem.

The Boolean Polynomial Systems Solving (Boolean PoSSo) Problem
Input: A Boolean polynomial set $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subseteq \mathbf{R}_2$.
Output: Solutions $(x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n$ s.t. $f_i(x_1, x_2, \ldots, x_n) = 0$ for $i = 1, 2, \ldots, m$.

Note that, the solutions of Boolean PoSSo problems are all restrained in the filed $\mathbb{F}_2^n$. Example 1 is a simple example for this problem, and we will use it to illustrate the procedures of our algorithm in the following parts of this paper.

Example 1. Given a Boolean polynomial system $\mathbf{P} = \{f_1, f_2, f_3\} = \{(x_4 + x_1x_2x_3 + x_1x_3)x_5 + (x_2 + 1)x_3x_4 + x_1x_2x_3 + x_2 + 1, (x_4 + x_1x_2x_3 + x_2 + 1)x_5 + (x_1x_2x_3 + x_1)x_4 + x_2x_3 + x_1, x_1x_3x_4x_5 + (x_1x_3 + x_2)x_4 + x_3 + x_2 + x_1\}$. The solutions of $\mathbf{P}$ for the Boolean PoSSo problem are (1, 0, 1, 0, 1), (1, 0, 1, 1, 1), (1, 1, 1, 1, 1), (0, 1, 0, 1, 0), (0, 0, 0, 1, 1).

In the following paragraphs of this paper, unless otherwise stated, for a polynomial system, we mean a Boolean polynomial system. Moreover, we use n to denote the number of variables and m to denote the number of polynomials.

Now we show how to use the characteristic algorithms to solve the Boolean PoSSo problem. First, we introduce some basic notions and notations about the characteristic set method over $\mathbb{F}_2$. For more details, the reader is referred to (Gao and Huang, 2012).

For a Boolean polynomial $P \in \mathbf{R}_2$, the *class* of P, denoted as $\mathrm{cls}(P)$, is the largest index c such that x_c occurs in P. If P is a constant, we set $\mathrm{cls}(P)$ to be 0. If $\mathrm{cls}(P) = c > 0$, we call x_c the *leading variable* of P, denoted as $\mathrm{lvar}(P)$. The leading coefficient of P as a univariate polynomial in $\mathrm{lvar}(P)$ is called the *initial* of P, and is denoted as $\mathrm{init}(P)$. Then, P can be written as $Ix_c + U$, where $I = \mathrm{init}(P), c = \mathrm{cls}(P)$, and U is a polynomial without variables x_c.

Given a polynomial system $\mathbf{P}$, we use $\mathrm{Zero}(\mathbf{P})$ to denote the solutions of this system, and we call $\mathrm{Zero}(\mathbf{P})$ the zero set of $\mathbf{P}$.

A sequence of nonzero polynomials

$$\mathcal{A}: \ A_1, A_2, \ldots, A_r$$

is a *triangular set* if either $r = 1$ and $A_1 = 1$, or $0 < \mathrm{cls}(A_1) < \cdots < \mathrm{cls}(A_r)$. A Boolean polynomial P is called *monic*, if $\mathrm{init}(P) = 1$. Moreover, if the elements of a triangular set are all monic, we call it a monic triangular set. Given a monic triangular set $\mathcal{A}: A_1, A_2, \ldots, A_r$, we can easily obtain its zero sets. Specifically speaking, for each evaluation of the variables in $\{x_1, x_2, \ldots, x_n\} \setminus \{\mathrm{lvar}(A_1), \mathrm{lvar}(A_2), \ldots, \mathrm{lvar}(A_r)\}$, we can obtain one solution in $\mathrm{Zero}(\mathcal{A})$ by recursive substitution. Hence, we have $|\mathrm{Zero}(\mathcal{A})| = 2^{n-r}$, and $n - r$ is called the *dimension* of $\mathcal{A}$.

The characteristic set algorithms proposed in (Gao and Huang, 2012) is to solve the following zero decomposition problem.

The Boolean Zero Decomposition Problem
Input: A Boolean polynomial set $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subseteq \mathbf{R}_2$.
Output: Monic triangular sets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_r$, such that $\mathrm{Zero}(\mathbf{P}) = \bigcup_i \mathrm{Zero}(\mathcal{A}_i)$, and $\mathrm{Zero}(\mathcal{A}_i) \cap \mathrm{Zero}(\mathcal{A}_j) = \emptyset$ for any $i \neq j$.

Obviously, if one can obtain such monic triangular sets $\mathcal{A}_i$, the solutions of $\mathbf{P}$ can be easily achieved, hence the corresponding Boolean PoSSo problem is solved. Moreover, $|\mathrm{Zero}(\mathbf{P})| = \sum_i 2^{n-r_i}$, where r_i is the number of polynomials in $\mathcal{A}_i$.

Example 2. For the polynomial system $\mathbf{P}$ in Example 1. By the characteristic set method, we can obtain five monic triangular sets $\mathcal{A}_1 = \{x_1 + 1, x_2, x_3 + x_2 + 1, x_4, x_5 + x_2 + 1\}$, $\mathcal{A}_2 = \{x_1 + 1, x_2, x_3 + 1, x_4 + 1,$

$x_5+1\}, \mathcal{A}_3 = \{x_1+1, x_2+1, x_3+x_2, x_4+1, x_5+1\}, \mathcal{A}_4 = \{x_1, x_2+1, x_3, x_4+1, x_5\}, \mathcal{A}_5 = \{x_1, x_2, x_3, x_4+1, x_5+1\}$, which are the outputs of the Boolean zero decomposition problem.[1]

3. An improved characteristic set algorithm

In this section, we will introduce the improved characteristic set algorithm **BCS**. First, we define a new kind of elimination operation. In the following of this paper, we use $\mathrm{tdeg}(P)$ to denote the total degree of a polynomial, and $\mathrm{lm}(P)$ to denote the leading monomial of P w.r.t. a monomial order (Cox et al., 1992).

Definition 3. Suppose P is a polynomial and $\mathbf{M}$ is a monic triangular set. Let P' and $\mathbf{M}'$ be the outputs of the following process.

Step 1. Set P' to be P and $\mathbf{M}'$ to be $\mathbf{M}$.
Step 2. While P' is monic and nonlinear, and $\exists Q \in \mathbf{M}'$ s.t. $\mathrm{cls}(P') = \mathrm{cls}(Q)$, do:
  Step 2.1. If $\mathrm{tdeg}(P') < \mathrm{tdeg}(Q)$, then replace Q with P' in $\mathbf{M}'$;
  Step 2.2. Set P' to be $P' + Q$.
Step 3. Output P' and $\mathbf{M}'$.

We call P' the **add-remainder** of P w.r.t. $\mathbf{M}$, denoted by $\mathrm{arem}(P, \mathbf{M})$. $\mathbf{M}'$ is called the **reducer sequence** of P w.r.t. $\mathbf{M}$, denoted by $\mathrm{rseq}(P, \mathbf{M})$.

Obviously, we have $\mathrm{Zero}(P, \mathbf{M}) = \mathrm{Zero}(\mathrm{arem}(P, \mathbf{M}), \mathrm{rseq}(P, \mathbf{M}))$. Now we show the precise process of **BCS**

Algorithm 1: BCS.

input : A polynomial system $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$.
output : Monic triangular sets $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_t\}$ such that $\mathrm{Zero}(\mathbf{P}) = \cup_{i=1}^{t} \mathrm{Zero}(\mathcal{A}_i)$ and $\mathrm{Zero}(\mathcal{A}_i) \cap \mathrm{Zero}(\mathcal{A}_j) = \emptyset$

```
1 P* ← {P}, A* ← ∅;
2 while P* ≠ ∅ do                                   /* P* is a group of polynomial sets */
3     Select and remove a polynomial set Q from P*;
4     Let A and Q* be the output of Triset(Q);
5     if A ≠ ∅ then A* ← A* ∪ {A};
6     P* ← P* ∪ Q*;
7 return A*.
```

In **BCS**, **Triset** is the sub-algorithm that solves the current polynomial system and generates some new polynomial systems, and it is the major part of **BCS**. Here we explain several main processes of **Triset**:

- At Step 3-13, we are trying to find linear polynomials and monic polynomials in the current system. If there is a linear polynomial $x_c + L$, we use L to substitute x_c in other polynomials, and move this $x_c + L$ into the monic triangular set $\mathcal{A}$. For the monic polynomials, we execute `AddReduce` to eliminate the leading variables of those polynomials which have the same classes by addition.
- At Step 16-21, we convert the chosen polynomial into monic polynomial by zero decomposition. It is based on the fact that $\mathrm{Zero}(Ix_c + U) = \mathrm{Zero}(x_c + U, I + 1) \cup \mathrm{Zero}(I, U)$. Then we compute $\mathrm{arem}(I, \mathbf{M})$ to simplify I. $\mathrm{arem}(I, \mathbf{M})$ is added into the new generated polynomial set. If $\mathrm{arem}(I, \mathbf{M})$ is monic and nonlinear, we add $\mathrm{arem}(I, \mathbf{M}) + 1$ into $\mathbf{M}$. Otherwise, we add $\mathrm{arem}(I, \mathbf{M}) + 1$ into $\mathbf{P}$.

[1] By different algorithms, the output triangular sets may be different.

Function: `Simplify`.

input : A polynomial set $\mathbf{P}$ and a monic polynomial set $\mathbf{M}$.
output : A linear triangular set $\mathcal{A}'$, and a nonlinear polynomial set $\mathbf{P}'$ and a monic polynomial set $\mathbf{M}'$, such that $\text{Zero}(\mathbf{P}) \cup \text{Zero}(\mathbf{M}) = \text{Zero}(\mathcal{A}') \cup \text{Zero}(\mathbf{P}') \cup \text{Zero}(\mathbf{M}')$.

$\mathcal{A}' \leftarrow \emptyset, \mathbf{P}' \leftarrow \mathbf{P}, \mathbf{M}' \leftarrow \mathbf{M}$;
if $1 \in \mathbf{P}'$ **then return** $\emptyset$, $\emptyset$ and $\emptyset$;
while $\mathbf{P}'$ *has a linear polynomial* $P = x_c + L$ **do** /* $\text{cls}(P) = c$ */
  $\mathbf{P}' = \mathbf{P}' \setminus \{P\}$;
  Substitute x_c with L for the other polynomials in $\mathbf{P}'$;
  Suppose $\mathbf{M}' = \{M_1, M_2, \ldots, M_k\}$;
  for $i \leftarrow 1$ *to* k **do**
    Substitute x_c with L in M_i, and obtain M_i';
    $\mathbf{M}' = \mathbf{M}' \setminus \{M_i\}$;
    if $\text{cls}(M_i') = \text{cls}(M_i)$ *and* M_i' *is not linear* **then** $\mathbf{M}' \leftarrow \mathbf{M}' \cup \{M_i'\}$;
    else $\mathbf{P}' = \mathbf{P}' \cup \{M_i'\}$;
  $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{x_c + L\}$;
  if $1 \in \mathbf{P}'$ **then return** $\emptyset$, $\emptyset$, and $\emptyset$;
return $\mathcal{A}'$, $\mathbf{P}'$ and $\mathbf{M}'$.

Function: `AddReduce`.

input : A monic polynomial set $\mathbf{P}$.
output : A polynomial set $\mathbf{P}'$, and a monic triangular set $\mathbf{M}'$, such that $\text{Zero}(\mathbf{P}) = \text{Zero}(\mathbf{P}') \cup \text{Zero}(\mathbf{M}')$.

$\mathbf{P}' \leftarrow \emptyset$;
repeat
  Sort the elements of $\mathbf{P}$ by classes and obtain polynomial sets $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_t$;
  /* The class of elements in $\mathbf{Q}_i$ is c_i */
  $\mathbf{M}' \leftarrow \emptyset$, $\mathbf{R} \leftarrow \emptyset$;
  for $i \leftarrow 1$ *to* t **do**
    Let Q be a polynomial in $\mathbf{Q}_i$, such that $\text{lm}(Q)$ is smallest w.r.t. a graded order ;
    $\mathbf{Q}_i \leftarrow \mathbf{Q}_i \setminus \{Q\}$, $\mathbf{M}' = \mathbf{M}' \cup \{Q\}$;
    while $\mathbf{Q}_i \neq \emptyset$ **do**
      Choose an element $Q_j \in \mathbf{Q}_i$, $\mathbf{Q}_i \leftarrow \mathbf{Q}_i \setminus \{Q_j\}$;
      $Q_j \leftarrow Q_j + Q$;
      if $Q_j = 1$ **then return** $\emptyset$ and $\emptyset$;
      if $Q_j \neq 0$, *and* Q_j *is linear or not monic* **then** $\mathbf{P}' \leftarrow \mathbf{P}' \cup \{Q_j\}$;
      if Q_j *is monic* **then** $\mathbf{R} \leftarrow \mathbf{R} \cup \{Q_j\}$;
  if $\mathbf{R} \neq \emptyset$ **then** $\mathbf{P} \leftarrow \mathbf{M}' \cup \mathbf{R}$;
until $\mathbf{R} = \emptyset$;
return $\mathbf{P}'$ and $\mathbf{M}'$.

- At Step 22-31, $\text{arem}(I, \mathbf{M})$ is equal to a constant $c \in \mathbb{F}_2$.[2] Obviously, we have $I + M_1 + M_2 + \cdots + M_k = c$ for some $M_1, \ldots, M_k \in \mathbf{M}$, then $\text{Zero}(Ix_c + U, \mathbf{M}) = \text{Zero}(cx_c + U, \mathbf{M})$. Hence, we replace $Ix_c + U$ with $cx_c + U$ in $\mathbf{P}$. Note that $\text{tdeg}(I) < \text{tdeg}(Ix_c + U)$, and we want to well use this polynomial with lower degree in the following process. However, when $\text{arem}(I, \mathbf{M})$ is constant, I is equivalent to a constant and we cannot achieve a polynomial with lower degree. Hence, by step 25-27, we generate a lower degree polynomial I' from I, then do the zero decomposition based on the cases of $\text{arem}(I', \mathbf{M}) = 0$ or 1.

As mentioned before, **BCS** is originated from the **MFCS** algorithm proposed in (Gao and Huang, 2012). The similarity of **BCS** and **MFCS** is the idea of using addition to eliminate variables. These two algorithms have four major differences:

[2] The probability of this case is extremely low from the observation in our experiments.

Algorithm 2: TriSet.

input : A polynomial system $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$.
output : A monic triangular set $\mathcal{A}$ and a group of polynomial sets $\mathbf{Q}^*$ such that $\text{Zero}(\mathbf{P}) = \text{Zero}(\mathcal{A}) \cup_{\mathbf{Q}\in\mathbf{Q}^*} \text{Zero}(\mathbf{Q})$, $\text{Zero}(\mathbf{Q}_i) \cap \text{Zero}(\mathcal{A}) = \emptyset$, $\text{Zero}(\mathbf{Q}_i) \cap \text{Zero}(\mathbf{Q}_j) = \emptyset$ for any $\mathbf{Q}_i, \mathbf{Q}_j \in \mathbf{Q}^*$ with $i \neq j$.

$\mathbf{Q}^* \leftarrow \emptyset$, $\mathcal{A} \leftarrow \emptyset$, $\mathbf{M} \leftarrow \emptyset$;
while $\mathbf{P} \neq \emptyset$ **do**
  repeat
    Let $\mathcal{A}'$, $\mathbf{P}'$, $\mathbf{M}'$ be the output of `Simplify`$(\mathbf{P}, \mathbf{M})$;
    if $\mathcal{A}', \mathbf{P}', \mathbf{M}' = \emptyset$ **then return** $\emptyset$ and $\mathbf{Q}^*$;
    else $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{A}'$, $\mathbf{P} \leftarrow \mathbf{P}'$, $\mathbf{M} \leftarrow \mathbf{M}'$;
    Let $\mathbf{M}'$ be the set of all monic polynomials in $\mathbf{P}$;
    $\mathbf{P} \leftarrow \mathbf{P} \setminus \mathbf{M}'$, $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{M}'$;
    if $\mathbf{M} \neq \emptyset$ **then**
      Let $\mathbf{M}'$ and $\mathbf{P}'$ be the output of `AddReduce`$(\mathbf{M})$;
      if $\mathbf{M}' = \emptyset$ **then return** $\emptyset$ and $\mathbf{Q}^*$;
      else $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{P}'$, and $\mathbf{M} \leftarrow \mathbf{M}'$;
  until $\mathbf{P}$ *doesn't contain linear polynomials*;
  if $\mathbf{P} \neq \emptyset$ **then** `/* The zero decomposition process */`
    $P \leftarrow$ `Choose`$(\mathbf{P})$, and suppose $\text{cls}(P) = k$, $P = Ix_k + U$; `/* Choose() is a function that chooses an element from a polynomial set */`
    if $\text{arem}(I, \mathbf{M})$ *is not a constant* **then**
      $I \leftarrow \text{arem}(I, \mathbf{M})$, $\mathbf{M} \leftarrow \text{rseq}(I, \mathbf{M})$;
      $\mathbf{P}_1 \leftarrow (\mathbf{P} \setminus \{P\}) \cup \mathcal{A} \cup \mathbf{M} \cup \{I, U\}$, and $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup \{\mathbf{P}_1\}$;
      $\mathbf{P} \leftarrow (\mathbf{P} \setminus \{P\}) \cup \{x_c + U\}$;
      if I *is monic and nonlinear* **then** $\mathbf{M} \leftarrow \mathbf{M} \cup \{I + 1\}$;
      else $\mathbf{P} \leftarrow \mathbf{P} \cup \{I + 1\}$;
    else
      if $\text{arem}(I, \mathbf{M}) = 1$ **then** $\mathbf{P} \leftarrow \mathbf{P} \setminus \{P\} \cup \{x_c + U\}$;
      else $\mathbf{P} \leftarrow \mathbf{P} \setminus \{P\} \cup \{U\}$;
      while $\text{arem}(I, \mathbf{M})$ *is a constant* **do**
        Suppose $I = x_{c_0} + x_{c_1} + \cdots + x_{c_k} + I'x_p + U'$, where $I' \neq 1$ and $c_0 > c_1 > \cdots > c_k > p > \text{cls}(U')$;
        $I \leftarrow I'$;
      $I \leftarrow \text{arem}(I, \mathbf{M})$, $\mathbf{M} \leftarrow \text{rseq}(I, \mathbf{M})$;
      $\mathbf{P}_1 \leftarrow \mathbf{P} \cup \mathcal{A} \cup \mathbf{M} \cup \{I\}$, and $\mathbf{Q}^* \leftarrow \mathbf{Q}^* \cup \{\mathbf{P}_1\}$;
      if I *is monic and nonlinear* **then** $\mathbf{M} \leftarrow \mathbf{M} \cup \{I + 1\}$;
      else $\mathbf{P} \leftarrow \mathbf{P} \cup \{I + 1\}$;
$\mathcal{A} \leftarrow \mathcal{A} \cup \mathbf{M}$, and **return** $\mathcal{A}$ and $\mathbf{Q}^*$.

1) In **MFCS**, `Choose` always chooses the polynomials with highest class. In **BCS**, `Choose` can be any form.
2) In **MFCS**, `AddReduce` is executed when the polynomials with the highest class are all monic. In **BCS**, `AddReduce` is executed when we have new generated monic polynomials.
3) In **BCS**, we add a new function `Simplify` to deal with the linear polynomials generated in the solving process.
4) In **BCS**, we use $\text{arem}(I, \mathbf{M})$ instead of I in the zero decomposition processes.

As we mentioned in Section 1, in the experiments of (Gao and Huang, 2012), some techniques were already added in the implementation of **MFCS**, here we denote this modified algorithm by **MFCS**$_1$. In **MFCS**$_1$, `Simplify` is used and `Choose` always chooses the polynomial whose initial is shortest, that is choosing the polynomial whose initial has the smallest number of monomials. Hence, **BCS** and **MFCS**$_1$ have the above differences 1), 2), 4).

From the complexity analysis in Section 5, we will see that these differences are important to reducing the complexity of characteristic set algorithms. Moreover, experimental results in Section 6 shows that by this modifications **BCS** is more efficient that **MFCS**$_1$ in practical computations.

Example 4. Let the polynomial system $\mathbf{P}$ in Example 1 be the input of **BCS**. We suppose that by `Choose` we always choose the polynomial with lowest degree. We show the procedure of **BCS**($\mathbf{P}$) step by step:

1. First, we let $\mathbf{P}$ be the input of **Triset**, and choose f_1 to do zero decomposition. The initial of f_1 is $I_1 = x_4 + x_1x_2x_3 + x_1x_3$. Then, after zero decomposition, $\mathbf{P}$ is updated to be $\mathbf{P}^1 = \{f_4, f_5, f_2, f_3\}$, where $f_4 = I_1 + 1$, $f_5 = x_5 + (x_2 + 1)x_3x_4 + x_1x_2x_3 + x_2 + 1$. We generate a new polynomial set $\mathbf{P}_1 = \{I_1, g_1, f_2, f_3\}$, where $g_1 = (x_2 + 1)x_3x_4 + x_1x_2x_3 + x_2 + 1$. Note that f_4 and f_5 is monic, hence $\mathbf{M} = \{f_4, f_5\}$. Now we choose f_2 to do zero decomposition. The initial of f_2 is $I_2 = x_4 + x_1x_2x_3 + x_2 + 1$, which is a monic polynomial. Thus, $\mathrm{arem}(I_2, \mathbf{M}) = I_2 + f_4 = x_1x_3 + x_2$, and $\mathrm{rseq}(I_2, \mathbf{M}) = \mathbf{M}$. We set I_2' to be $x_1x_3 + x_2$. Hence, $\mathbf{P}^1$ is updated to be $\mathbf{P}^2 = \{f_6, f_7, f_4, f_5, f_3\}$, where $f_6 = I_2' + 1$ and $f_7 = x_5 + (x_1x_2x_3 + x_1)x_4 + x_2x_3 + x_1$. We generate a new polynomial set $\mathbf{P}_2 = \{I_2', g_2, f_4, f_5, f_3\}$, where $g_2 = (x_1x_2x_3 + x_1)x_4 + x_2x_3 + x_1$. In $\mathbf{P}^2$, there are three monic polynomials f_7, f_4, f_5. Moreover, f_7 and f_5 have the same class. So after `AddReduce`, f_7 is reduced to $f_7' = f_7 + f_5 = (x_1x_2x_3 + x_2x_3 + x_3 + x_1)x_4 + (x_1x_2 + x_2)x_3 + x_2 + x_1 + 1$. Then, we choose $f_6 = x_1x_3 + x_2 + 1$ to do zero decomposition. Its initial is $I_3 = x_1$, hence $\mathbf{P}^2$ is updated to be $\mathbf{P}^3 = \{x_1 + 1, x_3 + x_2 + 1, f_7', f_4, f_5, f_3\}$, and we generate a new polynomial set $\mathbf{P}_3 = \{x_1, x_2 + 1, f_7', f_4, f_5, f_3\}$. Since $x_1 + 1$ and $x_3 + x_2 + 1$ are linear, we can execute `Simplify`, and after that, we obtain a linear triangular set $\mathcal{A}_1 = \{x_1 + 1, x_2, x_3 + x_2 + 1, x_4, x_5 + x_2 + 1\}$ and $\mathbf{P}^3$ becomes a empty set. Hence, **Triset** outputs $\mathcal{A}_1$ and $\{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$.
2. Let $\mathbf{P}_1 = \{I_1, g_1, f_2, f_3\}$ be the input of **Triset**. We choose g_1 to do zero decomposition. The initial of g_1 is $I_4 = (x_2 + 1)x_3$. Then, $\mathbf{P}_1$ is updated to be $\mathbf{P}_1^1 = \{p_1, p_2, I_1, f_2, f_3\}$, where $p_1 = I_4 + 1$ and $p_2 = x_4 + x_1x_2x_3 + x_2 + 1$. We generate a new polynomial set $\mathbf{P}_4 = \{I_4, p_3, I_1, f_2, f_3\}$, where $p_3 = x_1x_2x_3 + x_2 + 1$. In $\mathbf{P}_1^1$, we find that I_1 and p_2 are monic and have the same class. Thus by `AddReduce`, we obtain a new polynomial $p_4 = p_2 + I_1 = x_1x_3 + x_2 + 1$, and replace I_1 with p_4 in $\mathbf{P}_1^1$. Now $\mathbf{P}_1^1 = \{p_1, p_4, f_2, f_3\}$ and $\mathbf{M} = \{p_2\}$. We choose p_1 to continue the zero decomposition. The initial of p_1 is $I_5 = x_2 + 1$, hence $\mathbf{P}_1^1$ is updated to $\mathbf{P}_1^2 = \{x_2, x_3 + 1, p_4, f_2, f_3, p_2\}$, and we generate a new polynomial set $\mathbf{P}_5 = \{x_2 + 1, 1, p_4, f_2, f_3, p_2\}$. Now, we execute `Simplify` for $\mathbf{P}_1^2$. After that, the linear polynomial set $\mathcal{A}_2$ is $\{x_1 + 1, x_2, x_3 + 1, x_4 + 1, x_5 + 1\}$ and $\mathbf{P}_1^2 = \emptyset$. **Triset** outputs $\mathcal{A}_2$ and $\{\mathbf{P}_4, \mathbf{P}_5\}$.
3. Let $\mathbf{P}_2 = \{I_2', g_2, f_4, f_5, f_3\}$ be the input of **Triset**. We choose I_2' whose initial is $I_6 = x_1$ to do zero decomposition. Obviously, after zero decomposition, $\mathbf{P}_2$ is updated to be $\mathbf{P}_2^1 = \{x_1 + 1, x_3 + x_2, g_2, f_4, f_5, f_3\}$, and a new polynomial set $\mathbf{P}_6 = \{x_1, x_2, g_2, f_4, f_5, f_3\}$ is generated. Now we execute `Simplify` for $\mathbf{P}_2^1$. Then, we can obtain a linear triangular set $\mathcal{A}_3 = \{x_1 + 1, x_2 + 1, x_3 + x_2, x_4 + 1, x + 5 + 1\}$. Thus, the output of **Triset** is $\mathcal{A}$ and $\{\mathbf{P}_6\}$.
4. Let $\mathbf{P}_3 = \{x_1, x_2 + 1, f_7', f_4, f_5, f_3\}$ be the input of **Triset**. After `Simplify`, we have $\mathcal{A}_4 = \{x_1, x_2 + 1, x_3, x_4 + 1, x_5\}$ and $\mathbf{P}_3 = \emptyset$. Hence, **Triset** outputs $\mathcal{A}_4$ and $\emptyset$.
5. Let $\mathbf{P}_4 = \{I_4, p_3, I_1, f_2, f_3\}$ be the input of **Triset**. We choose I_4 to do zero decomposition. Its initial is $I_7 = x_2 + 1$. Then $\mathbf{P}_4$ is updated to be $\mathbf{P}_4^1 = \{x_2, x_3, p_3, I_1, f_2, f_3\}$, and we generate a new polynomial set $\mathbf{P}_7 = \{x_2 + 1, p_3, I_1, f_2, f_3\}$. After `Simplify`, p_3 is reduced to constant 1, thus **Triset** outputs $\emptyset$ and $\mathbf{P}_7$.
6. Let $\mathbf{P}_5 = \{x_2 + 1, 1, p_4, f_2, f_3, p_2\}$ be the input of **Triset**. Obviously, constant 1 is in $\mathbf{P}_5$, hence the output are two empty sets.
7. Let $\mathbf{P}_6 = \{x_1, x_2, g_2, f_4, f_5, f_3\}$ be the input of **Triset**. After `Simplify`, we have $\mathcal{A}_5 = \{x_1, x_2, x_3, x_4 + 1, x_5 + 1\}$, and **Triset** outputs $\mathcal{A}_5$ and $\emptyset$.
8. Let $\mathbf{P}_7 = \{x_2 + 1, p_3, I_1, f_2, f_3\}$ be the input of **Triset**. After `Simplify`, we have f_3 is reduced to constant 1, thus **Triset** outputs two empty sets.

Finally, **BCS** outputs five monic triangular sets $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5\}$.

4. The correctness of BCS

In this section, we will prove the correctness of **BCS**. In the following of this paper, when we say the first kind of zero decomposition, we mean the procedure of step 16-21 in **Triset**, and when we say the second kind of zero decomposition, we mean the procedure of step 22-31 in **Triset**.

Lemma 5. *Algorithm **Triset** is correct.*

Proof. It is easy to check that `AddReduce` and `Simplify` is correct. Let $\mathbf{P}_0$ be $\mathbf{P} \cup \mathcal{A} \cup \mathbf{M}$, which is the polynomial systems we deal with in Loop 2. Obviously, the elements in $\mathbf{P}_0$ may be updated after each iteration of Loop 2. Since $\text{Zero}(P, Q) = \text{Zero}(P, P + Q)$ and $\text{Zero}(Ix_c + U, x_c + L) = \text{Zero}(IL + U, x_c + L)$, we know that except the zero decomposition operations, other operations will not change the zero set of $\mathbf{P}_0$. Now we consider the first kind of zero decomposition, it is based on the fact that $\text{Zero}(\mathbf{P}_0) = \text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, x_c + U, I + 1) \cup \text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, I, U)$ and $\text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, x_c + U, I + 1) \cap \text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, I, U) = \emptyset$. Obviously, if we respectively use $\text{arem}(I, \mathbf{M})$, $\text{rseq}(I, \mathbf{M})$ instead of I, M, the above equations are still valid. Consider the second kind of zero decomposition. It happens when $\text{arem}(I, \mathbf{M})$ is a constant. Then $\text{Zero}(\mathbf{P}_0) = \text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, x_c + U)$ when $\text{arem}(I, \mathbf{M}) = 1$, and $\text{Zero}(\mathbf{P}_0) = \text{Zero}(\mathbf{P}_0 \setminus \{Ix_c + U\}, U)$ when $\text{arem}(I, \mathbf{M}) = 0$. By Step 25-27, a new non-constant polynomial I is generated. Then $\text{Zero}(\mathbf{P}_0) = \text{Zero}(\mathbf{P}_0, \text{arem}(I, M) + 1) \cup \text{Zero}(\mathbf{P}_0, \text{arem}(I, M))$, and $\text{Zero}(\mathbf{P}_0, \text{arem}(I, M) + 1) \cap \text{Zero}(\mathbf{P}_0, \text{arem}(I, M)) = \emptyset$. The above equations show that if the algorithm outputs the result, we have $\text{Zero}(\mathcal{A}) \cup_{\mathbf{Q} \in \mathbf{Q}^*} \text{Zero}(\mathbf{Q}^*) = \text{Zero}(\mathbf{P})$. This proves the correctness of the zero decomposition equations of the output.

Now we show that $\mathcal{A}$ must be a monic triangular set. There are two cases in which a polynomial P can be added into $\mathcal{A}$:

1. $P = x_c + L$ is a linear polynomial. It is monic, and we substitute all the x_c with L in other polynomials, hence other elements in $\mathcal{A}$ will not have x_c.
2. $P \in \mathbf{M}$ is added into $\mathcal{A}$ at step 32. In this case, for every class, there is only one element in $\mathbf{M}$ and it is monic. Therefore, the elements being added into $\mathcal{A}$ will have different class.

It implies that the elements in $\mathcal{A}$ are monic and have different classes, which means $\mathcal{A}$ is a monic triangular set.

Now let's prove the termination of **Triset**. It is sufficient to show that the loop of Step 2 terminates. We prove this by induction. If $n = 1$, the termination is obvious. Now we assume when $n \leq k$, Loop 2 terminates. When $n = k + 1$, we prove that if no contradiction occurs, which means we don't obtain constant 1, we will convert all the polynomials with class $k + 1$ into monic ones. Suppose the `Choose` function doesn't choose the polynomial with class $k + 1$, then we will always deal with the polynomials with k variables. According to the hypothesis, we will find contradiction or achieve a monic polynomial set with different classes from these polynomials. If no contradiction occurs, then the `Choose` function have to choose the polynomial with class $k+1$. The above procedure will repeat until all the polynomials with class $k + 1$ are converted into monic polynomials. Then, after executing `AddReduce`, $\mathbf{P}$ will have one polynomials with class $k + 1$, and it is monic. After that, we only need to deal with the polynomials with k variables, thus Loop 2 will terminate at last. □

Theorem 6. *Algorithm* **BCS** *is correct.*

Proof. According to Lemma 5, it is easy to check that if **BCS** terminates, the output is correct. Therefore, we only need to prove the termination of **BCS**.

For a polynomial set in $\mathbf{P}^*$, we assign an index $(t_n, r_n, t_{n-1}, r_{n-1}, \ldots, t_1, r_1)$, where t_i is the number of non-monic polynomials with class i in $\mathbf{P}^*$, and r_i is the number of polynomials with class i in $\mathbf{P}^*$. Then we order the indexes by lexicography. Now we will show that the index of any polynomial set in $\mathbf{Q}^*$ is strictly smaller than the index of $\mathbf{Q}$. Let $\mathbf{P}_0 = \mathbf{P} \cup \mathbf{M} \cup \mathcal{A}$ be the current polynomial set in **Triset**. It is sufficient to show that in **Triset**, the indexes of $\mathbf{P}_0$ will not increase after $\mathbf{P}_0$ being

updated, and the index of a new generated $\mathbf{P}_1$ is always smaller than that of $\mathbf{P}_0$ before zero decomposition.

We consider the following four kinds of operations:

1) We execute `AddReduce`. Consider the monic polynomials with highest class c. Obviously, after `AddReduce`, t_c will not be change, and r_c will decrease. Thus, the index of $\mathbf{P}_0$ will decrease.
2) We execute `Simplify`. Suppose we have a linear polynomial $x_c + l$. Note that only polynomials with class not less than c will be changed after substitution. For these polynomials, if some of them are converted into polynomials with lower classes, then the index of $\mathbf{P}_0$ will decrease. If the classes of them are not changed after substitution, then only t_k with $k > c$ may decrease when some non-monic polynomials are converted into monic ones. This implies that the index of $\mathbf{P}_0$ will not increase.
3) We choose a non-monic polynomial $Ix_c + U$, where $\mathrm{arem}(I, \mathbf{M})$ is not a constant, to do zero decomposition. Then we replace $Ix_c + U$ by $x_c + U$ in $\mathbf{P}_0$, add $\mathrm{arem}(I, \mathbf{M}) + 1$, whose class is lower than c, into $\mathbf{P}_0$, and replace $\mathbf{M}$ by $\mathrm{rseq}(I, \mathbf{M})$ in $\mathbf{P}_0$. Note that, replacing $\mathbf{M}$ by $\mathrm{rseq}(I, \mathbf{M})$ will not change the index of $\mathbf{P}_0$. Hence, in this case, t_c, the number of non-monic polynomial with class c, decrease by 1, then the index of $\mathbf{P}_0$ will decrease. Moreover, in the new generated polynomial set $\mathbf{P}_1$, $Ix_c + U$ is replaced by U and $\mathrm{arem}(I, \mathbf{M})$, hence the index of $\mathbf{P}_1$ is lower than that of $\mathbf{P}_0$ before zero decomposition.
4) We choose a non-monic polynomial $Ix_c + U$, where $\mathrm{arem}(I, \mathbf{M})$ is a constant, to do zero-decomposition. $Ix_c + U$ is replaced by $x_c + U$ or U, and a new polynomial whose class is lower than c is added into $\mathbf{P}_0$. Thus t_c or $r_c$decrease by 1, which means the index of $\mathbf{P}_0$ will decrease. For the new generated polynomial set $\mathbf{P}_1$, obviously its index is equal to the index of the updated $\mathbf{P}_0$, hence is lower than the index of $\mathbf{P}_0$ before zero decomposition.

It is easy to show that a strictly decreasing sequence of indexes must be finite. This proves the termination of **BCS**. □

5. The complexity of BCS

In this section, we will estimate the complexity of **BCS**. In order to do this, we introduce the concept of zero decomposition tree. We can generate *the zero decomposition tree* of **BCS** as follows:

I) First, let the root node to be the input polynomial system **P**, and let a pointer $\mathcal{M}$ point to this node. The depth of the root node is set to be 0.
II) In the process of **Triset**, when we generate a new polynomial system by zero decomposition, we generate a new node, and set it to be this new polynomial system. Then let this node be the right child of the node pointed by $\mathcal{M}$. After we update the current polynomial system after zero decomposition, we generate a new node, and set it to be the updated polynomial system. Then let this node to be the left child of the node pointed by $\mathcal{M}$. After this, we let $\mathcal{M}$ point to the left child.
III) After we finished **Triset** one time, we will select a new polynomial set **Q** from $\mathbf{P}^*$ at Step 3 of **BCS** and run **Triset** again. At this time, we let $\mathcal{M}$ point to the node corresponding to **Q** and execute the operations in II) again.

The following figure shows the zero decomposition tree of Example 4. The root node **P** is corresponding to the input polynomial system.

We call the external path of a zero decomposition tree a *solving branch*. It is obvious that a solving branch is corresponding to one execution of **Triset**. Moreover, in this branch there is a node which is corresponding to the input of **Triset**. Obviously, when the solving branch is corresponding to first execution of **Triset**, this node is the root node. Otherwise, this node is the last node which is the right child of some node on this path. Moreover, we call the *initial depth* of a solving branch to be the depth of this node. For example, for the path from **P** to $\mathbf{P}_2^1$ in Fig. 1, its initial depth is determined by the depth of node $\mathbf{P}_2$, which is equal to 2.

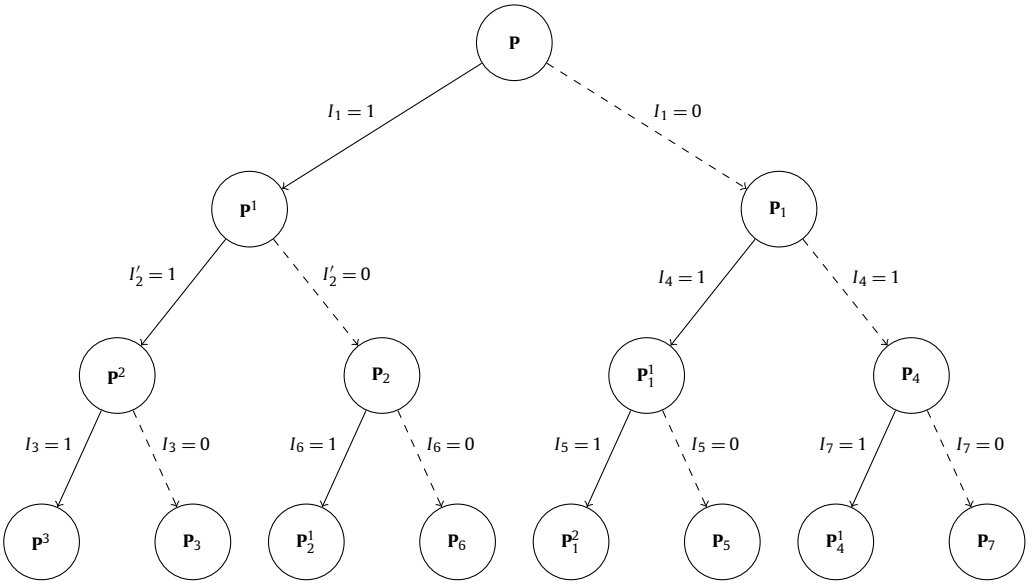

Fig. 1. The zero decomposition tree of Example 4.

Based on the zero decomposition tree, we can estimate the complexity of **BCS** by combining the complexity of **Triset** and the number of solving branches. Precisely speaking, if **Triset** has a complexity bound c and the depth of the zero decomposition tree is bounded by b, then the complexity of **BCS** is bounded by $c \cdot 2^b$.

5.1. The complexity of Triset

Lemma 7. *For a polynomial system* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *with* $\mathrm{tdeg}(f_i) \leq d$ *and a linear polynomial* $x_c + L$ *with class* c*. The bit-size complexity of the operation of substituting all* x_c *with* L *in* $\mathbf{P}$ *is* $O(dmn^{d+2}\log(n))$.

Proof. A polynomial f_i can be written as $P_1 x_c + P_2$. Then the substitution process of this polynomial is computing $P_1 L + P_2$. Since $\mathrm{tdeg}(f_i) \leq d$, we have $\mathrm{tdeg}(P_1) \leq d-1$ and $\mathrm{tdeg}(P_2) \leq d$. Moreover, we know that P_1 and P_2 have at most $n-1$ variables, thus P_1 has at most $\sum_{i=0}^{d-1}\binom{n-1}{i}$ terms and P_2 has at most $\sum_{i=0}^{d}\binom{n-1}{i}$ terms. L is a linear polynomial with at most $n-1$ terms. For computing $P_1 L$, we need to do at most $(n-1)\sum_{i=0}^{d-1}\binom{n-1}{i}$ times of monomial multiplication. For two monomial with $n-1$ variables, the multiplication is equal to the addition of two vectors with $n-1$ dimension, thus we need $n-1$ operations. Thus, we need $(n-1)^2\sum_{i=0}^{d-1}\binom{n-1}{i}$ operations to achieve the monomials of $P_1 L$. To compute $P_1 L$, we need to sum up all these monomials, and the complexity of this process is equal to the complexity of sorting all these monomials, which is $(n-1)((n-1)\sum_{i=0}^{d-1}\binom{n-1}{i})\log((n-1)\sum_{i=0}^{d-1}\binom{n-1}{i}) = O(dn^{d+2}\log(n))$.

Note that, $P_1 L$ and P_2 are two polynomials whose terms have been sorted, thus the complexity of $P_1 L$ plus P_2 is $2(n-1)\sum_{i=0}^{d}\binom{n-1}{i} = O(n^{d+1})$. Then, the complexity of computing $P_1 L + P_2$ is $O(dn^{d+2}\log(n)) + O(n^{d+1}) = O(dn^{d+2}\log(n))$. Hence, the complexity of m times of substitution is $O(dmn^{d+2}\log(n))$. □

In the following paragraphs, we define $\mathrm{tdeg}(\mathbf{P})$, the degree of a polynomial system $\mathbf{P}$, to be the highest total degree of the elements in $\mathbf{P}$.

Now we introduce the concept of backtracking for the polynomials occurring in the whole procedure of **BCS**. One can find that except adding $I+1$ into the current branch and I into the new generated branch, the purpose of other operations in **Triset** is replacing a polynomial P with another polynomial R. Precisely speaking, there are three kinds of operations:

(1) Replace $P = Ix_c + U$ with $R = x_c + U$ or U after zero decomposition.
(2) Replace $P = x_c + U_1$ with $R = U_1 + U_2$ after compute $P + P' = (x_c + U_1) + (x_c + U_2)$.
(3) Replace $P = g_1 x_k + g_2$ with $R = g_1 L + g_2$, after substituting x_k with the linear polynomial L in P.

Therefore, in the following, we say R can *backtrack* to P, if there is a polynomial sequence $P, P^1, P^2, \ldots, P^s, R$, s.t. P^1 replaced P, P^{i+1} replaced P^i and R replaced P^s by the above operations. Moreover, P can backtrack to itself.

Theorem 8. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$ *be the input polynomial system of* **BCS**, *and* $\mathrm{tdeg}(\mathbf{P}) = d$. *If the depths of branches in the zero decomposition tree are not greater than b, then the bit-size complexity of solving any branch of* **BCS** *is bounded by* $O(dm(m+b)\log(n)n^{d+3})$.

Proof. At first, we consider the first branch. The major operations in **Triset** are additions of two monic polynomials and substituting variables with linear polynomials. The complexity of other operations can be ignored when compared to these two kinds of operations. First, we consider the addition of two monic polynomials. Additions may occur in two cases. The first one is in `AddReduce` and the second one is when we compute add-remainder. Note that, after each time of zero decomposition, the number of polynomials in the current branch, that is the number of polynomials in $\mathbf{P} \cup \mathbf{M} \cup \mathcal{A}$, will increase by at most 1. Hence, after b times of zero decomposition, we will add b polynomials into the current branch. Then we can deduce that each polynomial occurring in $\mathbf{P} \cup \mathbf{M} \cup \mathcal{A}$ at any step of **Triset** can backtrack to one of the m input polynomials or one of the b newly added polynomials.

For any class c, consider the monic polynomials involved in `AddReduce`. Obviously, for these polynomials, additions can only be performed within the ones that cannot backtrack to the same polynomial. It implies that for any class c, one can execute at most $m+b$ times of addition in `AddReduce`. The bit-size complexity of adding two polynomials with degree d is $O(n^{d+1})$. Therefore, the bit-size complexity of the addition operations in `AddReduce` is bounded by $O((m+b)n^{d+2})$.

Now we consider the complexity of computing add-remainders. Actually, we need at most n times of addition for computing one add-remainder. In each time of zero decomposition, if $\mathrm{arem}(I, \mathbf{M})$ is not a constant, we need compute one add-remainder, and if $\mathrm{arem}(I, \mathbf{M})$ is a constant, we need compute at most $d-1$ add-remainders, since $\mathrm{tdeg}(I)$ decrease strictly. Hence, for b times of zero decomposition, we need at most $b(d-1)n$ times of addition. Therefore, the bit-size complexity of the addition operations in computing add-remainders is bounded by $O(b(d-1)n^{d+2})$.

When we execute the substitution in `Simplify` one time, we will eliminate one variable, thus `Simplify` can be executed at most n times. Each time, the number of polynomials that we need to do substitutions is at most $m+b$. Hence, according to Lemma 7, the complexity of executing `Simplify` n times is $O(dm(m+b)n^{d+3}\log(n))$.

In summary, the bit-size complexity of solving the first branch of the zero decomposition tree is bounded by $O(n^{d+2}(m+b+bd)) + O(dm(m+b)\log(n)n^{d+3}) = O(dm(m+b)\log(n)n^{d+3})$. Moreover, the above proof can be easily extended to other branches, since the polynomials in other branches polynomials can also backtrack to the input polynomials and the newly added polynomials. □

The results in the following subsection will show that when d is fixed, the value of b in Theorem 8 is polynomial w.r.t. n and m, which means that when d is fixed, solving one branch has polynomial-time complexity.

5.2. The complexity of **BCS**

In this section we will analyze the complexity of **BCS**, and our key problem is to estimate the bound of the depth for the solving branches in the zero decomposition tree. First, we consider a fundamental case, that is solving quadratic Boolean polynomial systems. Quadratic Boolean polynomial systems are typical nonlinear systems, and the problem of solving them is called the Boolean MQ problem.

Lemma 9. *Suppose the input of* **BCS** *is a quadratic polynomial system with n variables. Then the depth of the branches for the zero decomposition tree of* **BCS** *is less than n.*

Proof. Note that, each time we generate a new branch, we do zero decomposition for one time. We add $I+1$ in the current branch and I in the new generated branch, where $\mathrm{tdeg}(I) < \mathrm{tdeg}(P)$ and P is the chosen polynomial. Since the input system is quadratic and the degree of polynomials will not increase in the whole process of **BCS**, we have $\mathrm{tdeg}(P) = 2$. Therefore, $I+1$ and I are linear, which means in the current branch and the new generated branch, we both have a new linear polynomial. Then, by `Simplify`, we can eliminate one variable by the linear polynomial. Therefore, this can only happen at most $n-1$ times, which means the depth of the tree is less than n. □

Note that when solving quadratic system, branches with bigger initial depths have less variables, since more variables were eliminated in the former processes. Thus, the complexity of solving a branch with bigger initial depth is smaller. Based on this observation, we have the following lemma.

Lemma 10. *Suppose the input of* **BCS** *is a quadratic polynomial system with n variables and m polynomials. Let* $\mathbf{P}_1$ *be the system corresponding to a branch with initial depth* b_0 *and depth b. Then the bit-size complexity of* **Triset**$(\mathbf{P}_1)$ *is* $O((m+b)(n-b_0+1)^5\log(n-b_0+1)))$.

Proof. It is obvious that after one time of zero decomposition, the number of polynomials in this branch will increase at most by one. Therefore, at any step of **Triset**, the number of polynomials in this branch is always not bigger than $m+b$.

Since the initial depth of this branch is b_0, we have already eliminated b_0-1 variables before solving this branch. Thus, in the `Simplify` process, we can do the substitution $n-b_0+1$ times, and the complexity is $O((m+b)(n-b_0+1)^5\log(n-b_0+1))$ according to Lemma 7.

Now let us estimate the complexity of addition operations. The complexity of adding two quadratic polynomials with $n-b_0$ variables is $O((n-b_0)^3)$. For each class, the number of polynomials is not bigger than $m+b$, and the number of different classes is at most $n-b_0$. Therefore the number of additions is not bigger than $(n-b_0)(m+b)$, and the complexity of addition operations is $O((n-b_0)^4(m+b))$. Then, the complexity of **Triset** is $O((m+b)(n-b_0+1)^5\log(n-b_0+1))) + O((n-b_0)^4(m+b)) = O((m+b)(n-b_0+1)^5\log(n-b_0+1)))$. □

Lemma 11. *Let* $\mathbf{P}$ *be a quadratic polynomial system with n variables and m polynomials. If the depth of the branches of* **BCS**$(\mathbf{P})$ *are not bigger than b, then the bit-size complexity of* **BCS**$(\mathbf{P})$ *is bounded by*

1) $O((b+1)(m+b)2^{b-1}(n-b+1)^6)$, *when* $b < n-9.66$.
2) $O((b+1)(m+b)2^n)$, *when* $b \geq n-9.66$.

Proof. For a solving branch with depth b, except the root node there are b nodes in this branch. They can form a sequence $\{E_1, E_2, \ldots, E_b\}$, where $E_i = L$ or R, which means the i-th node is a left child or a right child respectively. For example, the node sequence of the first branch is $\{L, L, \ldots, L\}$. Then, the node sequence for a branch with initial depth $b_0 > 0$, must have the form

$$\{E_1, E_2, \ldots, E_{b_0-1}, R, \underbrace{L, L, \ldots, L}_{b-b_0}\},$$

where $E_i, 1 \le i \le b_0 - 1$ can be either L or R. Thus, the number of branches with initial depth b_0 is at most 2^{b_0-1}. According to Lemma 10, the bit-size complexity of **Triset** is $O((m+b)(n-b_0+1)^5\log(n-b_0+1)) \le O((m+b)(n-b_0+1)^6)$. Note that, there is only one branch with initial depth 0, and the complexity of solving this branch is bounded by $O((m+b)n^6)$. Then, the complexity of **BCS** is bounded by $(m+b)(n^6+\sum_{k=1}^{b} 2^{k-1}(n-k+1)^6)$. Function $2^{x-1}(n-x+1)^6$ reaches its maximal value when $x = n - 6/\log 2 + 1 = n - 9.66$. Thus, if $b < n - 9.66$, $(m+b)(n^6+\sum_{i=1}^{b} 2^{k-1}m(n-k+1)^6)) \le O((m+b)(b+1)2^{b-1}(n-b+1)^6)$. If $b \ge n - 9.66$, $(m+b)(n^6+\sum_{k=1}^{b} 2^{k-1}(n-k+1)^6 \le (m+b)(b+1)2^{n-9.66}(9.66)^8 = O((b+1)(m+b)2^n)$. □

Based on Lemma 9 and 11, we have the following theorem.

Theorem 12. *Let* $\mathbf{P}$ *be a quadratic polynomial system with n variables and m polynomials. The bit-size complexity of* **BCS**($\mathbf{P}$) *is bounded by* $O(n(m+n)2^n)$.

The bound $O(n(m+n)2^n)$ is the complexity bound of **BCS** in the worst case. To the best of the authors' knowledge, the best complexity result for solving Boolean quadratic polynomial system without side conditions is $4\log_2(n)2^n$ bit operations for the fast exhaustive search method proposed in (Bouillaguet et al., 2010). When some assumption is made for the system, the complexity of solving Boolean MQ problem can be less than $O(2^n)$. In (Bardet et al., 2013), Bardet et al. proposed an algorithm by combining exhaustive search and spare linear algebra, the complexity of this algorithm is $O(2^{0.841n})$, when $m = n$ under some precise algebraic assumptions which are satisfied with probability very close to 1. Moreover, a probabilistic variant of their algorithm (Las Vegas type) has expected complexity $O(2^{0.792n})$.

In the following, we consider the polynomial systems with degree higher than 2.

Proposition 13. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\}$ *be the input polynomial system of* **BCS**, *and* $tdeg(\mathbf{P}) = d$. *For any* `Choose`, *the depths of the branches in the zero decomposition tree are not bigger than* $m\sum_{j=1}^{d-1}\binom{n}{j}$.

Proof. It is sufficient to prove the theorem in the worst case, that is the input polynomials are all with class n, and when we choose a polynomial $P = Ix_c + U$ to do zero decomposition, the polynomial we add into $\mathbf{P}$ and the polynomial replacing P in $\mathbf{P}$ are all with class $c-1$.

We consider the first branch. Let M_c denote the set of polynomials with class c that can be backtracked by other polynomials. Moreover, if there are several polynomials with c which can be backtracked from the same polynomial with lower class, then only one of them, as a canonical element, is in M_c. Obviously $|M_n| \le m$, and the number of zero decomposition for polynomials with class n is not larger than m. Then, according to the proof of Theorem 8, polynomials in M_{n-1} can be grouped into the following two polynomial sets:

1. $\mathbf{P}_I$: the newly added polynomials.
2. $\mathbf{P}_R$: the polynomials which can backtrack to polynomials with class n.

Obviously, we have $|\mathbf{P}_I| \le M_n \le m$, $\text{tdeg}(\mathbf{P}_I) \le d-1$, $|\mathbf{P}_R| \le M_n \le m$ and $\text{tdeg}(\mathbf{P}_R) \le d$.

Similarly, the polynomials in M_{n-2} can be sorted into the following four polynomial sets:

1. $\mathbf{P}_{II}$: the new polynomials which are added when we choose polynomials in $\mathbf{P}_I$ to do zero decomposition.
2. $\mathbf{P}_{IR}$: the new polynomials which are added when we choose polynomials in $\mathbf{P}_R$ to do zero decomposition.
3. $\mathbf{P}_{RI}$: the polynomials which can backtrack to polynomials in $\mathbf{P}_I$.
4. $\mathbf{P}_{RR}$: the polynomials which can backtrack to polynomials in $\mathbf{P}_R$.

Then, we have $|\mathbf{P}_{JJ}|, |\mathbf{P}_{RR}|, |\mathbf{P}_{JR}|, |\mathbf{P}_{RJ}| \le m$. $\text{tdeg}(\mathbf{P}_{JJ}) \le d-2$, $\text{tdeg}(\mathbf{P}_{JR}) \le d-1$, $\text{tdeg}(\mathbf{P}_{RJ}) \le d-1$ and $\text{tdeg}(\mathbf{P}_{RR}) \le d$.

Recursively, we can define $\mathbf{P}_{O_1O_2\cdots O_k}$, where O_i is I or R, and the polynomials in M_{n-k} can be sorted into these polynomial sets. We have $|\mathbf{P}_{O_1O_2\cdots O_k}| \leq m$ and $\mathrm{tdeg}(\mathbf{P}_{O_1O_2\cdots O_k}) \leq d-s$, where s is number of I in these subscripts O_i. When we choose a polynomial to do zero decomposition, its total degree must be higher than 1. Therefore, for class $n-k$, when we do zero decomposition, we can only choose the polynomials in such $\mathbf{P}_{O_1O_2\cdots O_k}$ that the number of I occurring in $O_1, O_2, \ldots, O_k$ is at most $d-2$. It means that the number of zero decompositions for class $n-k$ is not larger than $m(\sum_{i=0}^{d-2}\binom{k}{i})$. Hence, the total number of zero decompositions is bounded by $m\sum_{k=0}^{n-1}(\sum_{i=0}^{d-2}\binom{k}{i}) = m\sum_{j=1}^{d-1}\binom{n}{j}$.

It is easy to see that the above bound can be easily extended to other branches, since we only need the property that when each time we do zero decomposition only one new polynomial with degree lower than the degree of the chosen polynomial is added into the current branch, which is satisfied for any branches. □

Remark 1. Note that the properties we used in the proof of Proposition 13 are also valid for the **MFCS** algorithm proposed in (Gao and Huang, 2012). Hence this depth bound is also valid for **MFCS**.

The above proposition shows that by any choose function, the depth of the zero decomposition tree is bounded by $m\sum_{j=1}^{d-1}\binom{n}{j}$. Actually, the depth can be much smaller, when we use some specific `Choose` functions. In the following, we consider the following choose function.

`Choose`$_1$: Choose a polynomial P from a polynomial set $\mathbf{P}$ s.t. $\mathrm{tdeg}(P) = min_{f\in\mathbf{P}}\mathrm{tdeg}(f)$.

Now we estimate the complexity of **BCS** with `Choose`$_1$ as the choose function. First, we prove the following lemma.

Lemma 14. *Let* $\mathbf{P}$ *be a polynomial system with n variables. Suppose in some step of* **Triset**($\mathbf{P}$)*, the polynomial set* $\mathcal{A}\cup\mathbf{M}$ *has n elements with different classes, then* **Triset** *will terminate without doing zero decomposition in the following steps.*

Proof. From the definition of $\mathcal{A}$ and $\mathbf{M}$, we know that $\mathcal{A}$ and $\mathbf{M}$ contain some monic polynomials. Note that in **Triset**, we execute `Simplify` after we generated a new linear polynomial. From the assumption, we know that the classes of the elements in $\mathbf{M}$ are different from those in $\mathcal{A}$, hence $\mathcal{A}\cup\mathbf{M}$ forms a monic triangular set. Obviously, the element in $\mathcal{A}\cup\mathbf{M}$ with the lowest class will have the form x_1+b, where $b=0$ or 1. Then, after `Simplify` the element with class 2 will also have the form x_2+b, where $b=0$ or 1. Recursively, all the elements will have the form x_c+b after `Simplify`, which means the values of the variables are fixed, then other polynomials in $\mathbf{P}$ will be converted into constant after `Simplify`, hence no more zero decomposition is needed. □

Proposition 15. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *be a polynomial system with degree d, and* `Choose`$_1$ *be the choose function of* **Triset**, *then* **Triset**($\mathbf{P}$) *will terminate after* $(2d-3)n$ *times of zero decomposition.*

Proof. For the polynomial sets $\mathbf{P}$, $\mathbf{M}$, $\mathcal{A}$ at any step of **Triset**, we can define an index vector $\mathcal{T} = (\mathbf{d_0}, \mathbf{d_1}, \ldots, \mathbf{d_n})$ as follows. For $\mathbf{d}_0$, we have:

1) If there is a linear polynomial in $\mathbf{P}$, set $\mathbf{d}_0$ to be 1.
2) If there are non-monic polynomials in $\mathbf{P}$, set $\mathbf{d}_0$ to be the lowest total degree for these non-monic polynomials.
3) If all polynomials in $\mathbf{P}$ are monic, set $\mathbf{d}_0$ to be d. If $\mathbf{P}=\emptyset$, set $\mathbf{d}_0$ to be 0.

For $\mathbf{d}_i$, $1 \leq i \leq n$, we have:

1) If the classes of monic polynomials in $\mathcal{A}\cup\mathbf{M}$ are not equal to i, then set $\mathbf{d}_i$ to be d.
2) If there is a linear polynomial with class i in $\mathcal{A}$, then set $\mathbf{d}_i$ to be 1.
3) If there is a monic polynomial M with class i in $\mathbf{M}$, we set $\mathbf{d}_i$ to be $\mathrm{tdeg}(M)$.

In the following, we show that after zero decomposition, the sum of all entries in $\mathcal{T}$, denoted by $Sum(\mathcal{T})$, will strictly decrease. Note that at Step 14 the polynomials in $\mathbf{P}$ are not monic. Hence $\mathbf{d}_0 = \mathrm{tdeg}(P)$, where P is the chosen polynomial. Suppose $\mathrm{init}(P) = I$. Then, there are five cases:

1) I is not monic, hence $\mathrm{arem}(I, \mathbf{M}) = I$. Since $\mathrm{tdeg}(I) < \mathrm{tdeg}(P)$, and P is the non-monic polynomial in $\mathbf{P}$ with lowest degree, then after adding $I+1$ into $\mathbf{P}$, the value of d_0 will at least decrease by 1. Moreover, the value of other elements in $\mathcal{T}$ will not increase. Thus $Sum(T)$ will strictly decrease.
2) I is linear, then $\mathrm{arem}(I, \mathbf{M}) = I$. Obviously, d_0 is equal to 1 after adding $I+1$ into $\mathbf{P}$, and the value of other elements in $\mathcal{T}$ will not increase. Thus $Sum(\mathcal{T})$ will strictly decrease.
3) I is monic, nonlinear, and $\mathrm{arem}(I, M) = I$. It means that the classes of the polynomials in $\mathbf{M}$ are not equal to $c = \mathrm{cls}(I)$, hence before zero decomposition, $\mathbf{d}_c$ is d. It implies that before zero decomposition, $\mathbf{d}_0 + \mathbf{d}_c = \mathrm{tdeg}(P) + d$. Since $\mathrm{tdeg}(I) < \mathrm{tdeg}(P)$, then after adding $I+1$ into $\mathbf{M}$, we have $\mathbf{d}_0 \le d$, $\mathbf{d}_c = \mathrm{tdeg}(I+1) \le \mathrm{tdeg}(P) - 1$, which implies $\mathbf{d}_0 + \mathbf{d}_c \le d + \mathrm{tdeg}(P) - 1$. Moreover, the value of other elements in $\mathcal{T}$ will not increase, hence $Sum(\mathcal{T})$ will strictly decrease.
4) $\mathrm{arem}(I, \mathbf{M}) \neq I$ and $\mathrm{arem}(I, \mathbf{M})$ is not a constant. It means that I is monic, and there are some polynomials $M_1, M_2, \ldots, M_k$ in $\mathbf{M}$ such that $I + M_1 + M_2 + \cdots + M_k = \mathrm{arem}(I, \mathbf{M})$. Suppose the polynomials in $\mathrm{rseq}(I, \mathbf{M})$ are $M_1', M_2', \ldots, M_k'$, and $\mathrm{cls}(M_i) = \mathrm{cls}(M_i') = c_i$. Note that, for two polynomials P and Q, $\mathrm{tdeg}(P) + \mathrm{tdeg}(Q) \ge \mathrm{tdeg}(P+Q) + \mathrm{tdeg}(Q)$, if $\mathrm{tdeg}(Q) \le \mathrm{tdeg}(P)$. Hence, we can deduce that $\mathrm{tdeg}(\mathrm{arem}(I, \mathbf{M})) + \mathrm{tdeg}(M_1') + \cdots + \mathrm{tdeg}(M_k') \le \mathrm{tdeg}(I) + \mathrm{tdeg}(M_1) + \cdots + \mathrm{tdeg}(M_k) < \mathrm{tdeg}(P) + \mathrm{tdeg}(M_1) + \cdots + \mathrm{tdeg}(M_k)$. If $R = \mathrm{arem}(I, \mathbf{M})$ is monic and nonlinear, suppose $\mathrm{cls}(R) = c$, then we can deduce that the value of $\mathbf{d}_0 + \mathbf{d}_c + \mathbf{d}_{c_1} + \cdots + \mathbf{d}_{c_k}$ decrease strictly. Otherwise, it is easy to see that the value of $\mathbf{d}_0 + \mathbf{d}_{c_1} + \cdots + \mathbf{d}_{c_k}$ decrease strictly. It implies that $Sum(\mathcal{T})$ will strictly decrease.
5) $\mathrm{arem}(I, \mathbf{M})$ is a constant. By Step 25-27, we generate a new polynomial I' such that $\mathrm{tdeg}(I') < \mathrm{tdeg}(I)$ and $\mathrm{arem}(I', \mathbf{M})$ is not a constant. Then we set I to be I', add $\mathrm{arem}(I, \mathbf{M})$ into $\mathbf{P}$ and update $\mathbf{M}$ by $\mathrm{rseq}(I, \mathbf{M})$. It is obvious that one of the above four cases will occur, thus $Sum(\mathcal{T})$ will strictly decrease.

In all, for any cases, $Sum(\mathcal{T})$ will strictly decrease after zero decomposition. Now we consider the variation of $Sum(\mathcal{T})$ after `AddReduce` and `Simplify`. In `AddReduce`, we compute the addition of two nonlinear and monic polynomials Q_1 and Q_2, where $\mathrm{cls}(Q_1) = \mathrm{cls}(Q_2) = c$, and keep the one with lowest degree in $\mathbf{M}$. From the definition of $\mathbf{d}_i$, we know that $\mathbf{d}_c$ will not increase after addition, and obviously $\mathbf{d}_0$ will not increase. Therefore, we can conclude that after `AddReduce`, $Sum(\mathcal{T})$ will not increase.

Now we consider `Simplify`. In `Simplify`, Loop 3 may iterate several times. We focus on the first time. Since there are several linear polynomials in $\mathbf{P}$, we have $\mathbf{d}_0 = 1$. We consider the one with lowest class in these linear polynomials, and denote it by $L = x_c + l$, where c is the class of L. Then, there are two cases:

1) Before `Simplify`, the classes of polynomials in $\mathbf{M}$ are not equal to c, then $\mathbf{d}_c = d, \mathbf{d}_0 = 1$. After substituting x_c with l and add L into $\mathcal{A}$, we have $\mathbf{d}_c = 1, \mathbf{d}_0 \le d$. It is easy to see that after each iteration of Loop 3, $\mathbf{d}_i$ will not increase. Hence, $Sum(T)$ doesn't increase after `Simplify`.
2) Before `Simplify`, there is a polynomial Q with degree d' in $\mathbf{M}$ such that $\mathrm{cls}(Q) = \mathrm{cls}(L) = c$, then $\mathbf{d}_c + \mathbf{d}_0 = 1 + d'$. After substituting x_c with l, we will achieve a nonlinear polynomial $Q + L$ with $\mathrm{tdeg}(Q + L) \le d'$.
   - If $Q + L$ is not monic, then $\mathbf{d}_0 \le d'$ after adding $Q + L$ into $\mathbf{P}$. Note that $\mathrm{cls}(Q + L) < \mathrm{cls}(L)$, thus the classes of other linear polynomials occurring in `Simplify` will be bigger than $\mathrm{cls}(Q + L)$, which means $Q + L$ will not be changed in the following substitutions. Hence, after `Simplify`, $Q + L$ is still in $\mathbf{P}$, then $\mathbf{d}_0 \le d'$, $\mathbf{d}_c = 1$. Therefore, $Sum(\mathcal{T})$ doesn't increase after `Simplify`.
   - If $Q + L$ is a monic polynomial, and before `Simplify` the polynomials in $\mathbf{M}$ have classes different from $c' = \mathrm{cls}(Q + L)$, then we have $\mathbf{d}_{c'} = d, \mathbf{d}_0 = 1, \mathbf{d}_c = d'$ before `Simplify`. Since $Q + L$ is a monic polynomial and it will not be changed by the following substitutions, this polynomial or another polynomial with same class and lower total degree will be added into $\mathbf{M}$ after

the following AddReduce process. Therefore, after Simplify and the following AddReduce, $\mathbf{d}_0 \leq d, \mathbf{d}_c = 1, \mathbf{d}_{c'} \leq d'$, which means $Sum(\mathcal{T})$ will not increase.

- If $Q + L$ is a monic polynomial and $c' = \mathrm{cls}(Q + L) = \mathrm{cls}(M_j)$ for some $M_j \in \mathbf{M}$, then $Q + L$ may become 0 after the following AddReduce. In this case, after Simplify and the following AddReduce, $\mathbf{d}_0$ may become d, and $\mathbf{d}_{c'}$ may not decrease, which means $Sum(T)$ may increase. In the worst cases that $\mathbf{d}_c = 2$ before Simplify, $Sum(\mathcal{T})$ will increase at most $(d+1) - (1+2) = d-2$.

We know that when $\mathcal{T} = (d, d, \ldots, d)$, $Sum(\mathcal{T})$ reaches its maximal value $d(n+1)$. Suppose we have executed zero decomposition for $(2d-3)n$ times. Note that for $Sum(\mathcal{T})$, the increase cases only happen after Simplify was executed, thus it can occur at most n times. It means that $Sum(\mathcal{T})$ can increase at most by $n(d-2)$. Thus, after $(d-1)n + (d-2)n$ times of zero decomposition, we have $Sum(\mathcal{T}) \leq d(n+1) + n(d-2) - (d-1)n - (d-2)n = n + d$. Now we show that when $Sum(\mathcal{T}) = d + n$, **Triset** will terminate without further zero decomposition. There are two cases for $Sum(\mathcal{T}) = d + n$. The first case is that $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n < d$, which means for any $1 \leq i \leq n$, there is a monic polynomial with class i in $\mathcal{A} \cup \mathbf{M}$. According to Lemma 14, **Triset** will terminate without further zero decomposition. The second case is that $\mathbf{d}_j = d$ for some $1 \leq j \leq n$. Since $Sum(\mathcal{T}) = d + n$, we have $\mathbf{d}_0 = 1, \mathbf{d}_1 = 1, \ldots, \mathbf{d}_{j-1} = 1, \mathbf{d}_j = 1, \ldots, \mathbf{d}_n = 1$. Obviously, $x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$ will not occur in polynomials in $\mathbf{P}$. Hence the polynomials in $\mathbf{P}$ with lowest degree have degree 1 and have leading variable x_j, which implies **Triset** will terminate without further zero decomposition. It is easy to see that if $Sum(\mathcal{T}) < d + n + 1$, **Triset** will also terminate without further zero decomposition. □

The above proposition shows that the first branch of the zero decomposition tree has depth not bigger than $(2d-3)n$. Note that, in the above proof, the critical property we used is $\mathrm{tdeg}(I+1) < \mathrm{tdeg}(P)$. For other branches generated by considering $I = 0$, we also have $\mathrm{tdeg}(I) = \mathrm{tdeg}(I+1) < \mathrm{tdeg}(P)$, hence same result can be proved. Consequently, we have the following theorem.

Theorem 16. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *with* $\mathrm{tdeg}(\mathbf{P}) = d$ *be the input of* **BCS**, *and* $\mathtt{Choose}_1$ *be the choose function. Then the depths of the branches of the zero decomposition tree are not bigger than* $(2d-3)n$.

By combining Theorem 8 and 16, we have the following complexity bound about **BCS**.

Theorem 17. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *with* $\mathrm{tdeg}(\mathbf{P}) = d$ *be the input of* **BCS**, *and* $\mathtt{Choose}_1$ *be the choose function. Then the complexity of* **BCS** *is bounded by* $O\left(d(m^2 + (2d-3)nm)\log(n) \times n^{d+3}2^{(2d-3)n}\right)$.

Furthermore, we consider the following choose function.

$\mathtt{Choose}_2$: Choose a polynomial P from a polynomial set $\mathbf{P}$ such that $\mathrm{tdeg}(\mathrm{init}(P)) = min_{f \in \mathbf{P}}\mathrm{tdeg}(\mathrm{init}(f))$

The following theorem shows that for $\mathtt{Choose}_2$ the above complexity bound for **BCS** is still valid.

Theorem 18. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *with* $\mathrm{tdeg}(\mathbf{P}) = d$ *be the input of* **BCS**, *and* $\mathtt{Choose}_2$ *be the choose function. Then the complexity of* **BCS** *is bounded by* $O\left(d(m^2 + (2d-3)nm)\log(n) \times n^{d+3}2^{(2d-3)n}\right)$.

Proof. It is sufficient to show that the depth of the zero decomposition tree is not larger than $(2d-3)n$. First, we consider the first branch. We define an new index vector $\mathcal{T}' = (\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_n)$ for the polynomial sets $\mathbf{P}$, $\mathbf{M}$, $\mathcal{A}$ in **Triset**, where the definitions of $\mathbf{d}_1, \ldots, \mathbf{d}_n$ are same as those of $\mathcal{T}$ in the proof of Proposition 16, and $\mathbf{d}_0$ is defined as follows:

- If there is a linear polynomial in $\mathbf{P}$, set $\mathbf{d}_0$ to be 0.
- If there are non-monic polynomials in $\mathbf{P}$, set $\mathbf{d}_0$ to be the lowest total degree of the initials of these non-monic polynomials.
- If all polynomials in $\mathbf{P}$ are monic, set $\mathbf{d}_0$ to be $d-1$. If $\mathbf{P}=\emptyset$, set $\mathbf{d}_0$ to be 0.

Similarly, $Sum(\mathcal{T}')$ is defined to be the sum of the entries in $\mathcal{T}'$. Obviously, we can prove that after different operations the variation of $Sum(T')$ is same as that of $Sum(T)$. Consider the maximal possible value of $Sum(\mathcal{T}')$ which is equal to $(n+1)d-1$ and achieved when $\mathcal{T}'=\{d-1,d,d,\ldots,d\}$. Then, after do zero decomposition for $(2d-3)n$ times, $Sum(T')$ is at most $d+n-1$. Then, we have either $\mathbf{d}_1,\mathbf{d}_2,\ldots,\mathbf{d}_n<d$, or $\mathbf{d}_0=0,\mathbf{d}_k=d,\mathbf{d}_i=1$, for some $1\leq k\leq n$ and any $1\leq i\leq n, i\neq k$. For these two cases, we can deduce that **Triset** will end without further zero decomposition. This implies that the depth of the first branch is not larger than $(2d-3)n$. Moreover, it is easy to see that this depth bound is still valid for other branches. □

Remark 2. For a polynomial set $\mathbf{P}$, we have $min_{f\in\mathbf{P}}\mathrm{tdeg}(\mathrm{init}(f))\leq min_{f\in\mathbf{P}}\mathrm{tdeg}(f)$. It means that compared with choosing the polynomial with lowest total degree, choosing the polynomial whose initial has lowest total degree may make $\mathbf{d}_0$ decrease faster, and this can induce a lower experimental complexity.

5.3. A variant algorithm with lower complexity bound

In this section, we show that by slightly modifying **BCS**, the case of $Sum(\mathcal{T})$ (or $Sum(\mathcal{T}')$) increasing after `Simplify` can be absolutely avoided.

We perform the following modification on **BCS**, and name the new algorithm $\mathbf{BCS}_g$:

Suppose $\mathbf{M}$ is changed after zero decomposition, `AddReduce` or `Simplify`. We sort $\mathbf{M}$, such that the former elements have the lower classes, and suppose $\mathbf{M}=\{M_1,M_2,\ldots,M_t\}$ after sorting. Then, sort the different monomials in $\mathbf{M}$ with respect to a graded order, and set the different nonlinear monomials as different new variables $y_1,y_2,\ldots,y_k$. Then each polynomial $M_i\in\mathbf{M}$ is a linear polynomial in variables $y_1,\ldots,y_k$ and $x_1,\ldots,x_n$. We generate the coefficient matrix $\mathcal{M}$ about $\mathbf{M}$, such that $\mathcal{M}(y_1,\ldots,y_k,x_1,\ldots,x_n)^T=(M_1,M_2,\ldots,M_t)^T$. Perform one-direction Gaussian elimination on $\mathcal{M}$, that is we only use the upper rows to eliminate the lower rows, and don't swap the position of two rows. Suppose, after Gaussian elimination, $\mathcal{M}$ becomes $\mathcal{M}'$. Then set $\mathbf{M}$ to be $\mathcal{M}'(y_1,\ldots,y_k,x_1,\ldots,x_n)^T$. If there are linear polynomials in $\mathbf{M}$, then repeat the following operations until there is no linear polynomial in $\mathbf{M}$:

Step 1. For each linear polynomial $L=x_c+l\in\mathbf{M}$, move it from $\mathbf{M}$ into $\mathcal{A}$;
Step 2. Substitute x_c with l for polynomials in $\mathbf{P}\cup\mathbf{M}$.

In the following we estimate the complexity of $\mathbf{BCS}_g$. First, we analyze the variation of $Sum(T)$ after zero decomposition. Evidently, for $\mathcal{M}$, Gaussian elimination does not increase the total degree of the polynomial corresponding to each row, since the monomials are sorted with decreasing degree. Hence, if no linear polynomial is generated after Gaussian elimination, $\mathbf{d}_0$ will not change and $\mathbf{d}_1,\ldots,\mathbf{d}_n$ will not increase, thus $Sum(\mathcal{T})$ will not increase. Hence, if not linear polynomial is generated after Gaussian elimination, $Sum(\mathcal{T})$ will strictly decrease. In the following, we consider the case that new linear polynomials are generated after Gaussian elimination:

- Suppose we add a non-monic polynomial $\mathrm{arem}(I,\mathbf{M})$ into $\mathbf{P}$ after zero decomposition, then $\mathbf{M}$ may be updated by $\mathrm{rseq}(I,\mathbf{M})$, after computing $\mathrm{arem}(I,\mathbf{M})$. Since $\mathrm{cls}(\mathrm{arem}(I,\mathbf{M}))$ is smaller than the classes of polynomials in $\mathrm{rseq}(I,\mathbf{M})$, hence is smaller than the classes of the new generated linear polynomials by Gaussian elimination. Thus, $\mathrm{arem}(I,\mathbf{M})$ will not be changed by substitutions w.r.t. these new linear polynomials, thus $Sum(\mathcal{T})$ will strictly decrease in this case.
- Suppose we add a monic polynomial in $\mathbf{M}$ after zero decomposition. Note that, in the proof of Proposition 16, we assume that $\mathbf{d}_0\leq d$ after zero decomposition, and this still holds after substitutions were executed, hence $Sum(\mathcal{T})$ will strictly decrease.

Now we consider `AddReduce`. After `AddReduce`, suppose there is a monic polynomial M in $\mathbf{M}$. If before `AddReduce`, there is no polynomial with class equal to $\mathrm{cls}(M)$ in $\mathbf{M}$, we call this M a *totally new* polynomial in $\mathbf{M}$. If before `AddReduce`, there is a polynomial M' with $\mathrm{cls}(M') = \mathrm{cls}(M)$ and $\mathrm{tdeg}(M') > \mathrm{tdeg}(M)$, then we say M' is replaced by M in $\mathbf{M}$. Similarly as above, if no linear polynomials is generated by Gaussian elimination, $\mathbf{d}_0$ will not be changed and $\mathbf{d}_i$ with $1 \leq i \leq n$ will not increase, hence $Sum(\mathcal{T})$ will not increase. Now suppose there are several linear polynomials generated by Gaussian elimination, and P_0 is the one with lowest class, where $\mathrm{cls}(P_0) = c$. For $\mathbf{M}$ and $\mathbf{P}$ before `AddReduce`, let $R_1 = \{P : P \in \mathbf{M}, \mathrm{cls}(P) \leq c\} \cup \{P : P \in \mathbf{P}, \mathrm{cls}(P) < c\}$, and $S_1 = \{\mathrm{lm}(f) : f \in R_1\}$. For $\mathbf{M}$ and $\mathbf{P}$ after Gaussian elimination, let $R_2 = \{P : P \in \mathbf{M}, \mathrm{cls}(P) \leq c\} \cup \{P : P \in \mathbf{P}, \mathrm{cls}(P) < c\}$, and $S_2 = \{\mathrm{lm}(f) : f \in R_2\}$. Here $\mathrm{lm}(f)$ is the leading monomial of f w.r.t. the graded order used in the Gaussian elimination process. Note that, in `AddReduce` or Gaussian elimination, after we compute the addition of two polynomials f and g with $\mathrm{lm}(f) \leq \mathrm{lm}(g)$, f is not changed and g is converted into $g' = f + g$, hence $\{\mathrm{lm}(g), \mathrm{lm}(f)\} \subset \{\mathrm{lm}(g'), \mathrm{lm}(f)\}$. Therefore, we can deduce that $S_1 \subset S_2$. Then there are two cases:

1) Before `AddReduce`, there is no polynomial with class c in $\mathbf{M}$. Then before `AddReduce`, we have $\mathbf{d}_c = d, \mathbf{d}_0 \geq 2$. After Gaussian elimination and the following substitutions, we have $\mathbf{d}_c = 1, \mathbf{d}_0 \leq d$. Since other $\mathbf{d}_i$ will not increase, we have $Sum(\mathcal{T})$ will decrease.
2) Before `AddReduce`, there is a polynomial Q_0 with class c in $\mathbf{M}$. Since after Gaussian elimination, $\mathrm{lm}(P_0) \neq \mathrm{lm}(Q_0)$ and $\mathrm{lm}(Q_0) \in S_2$, then $\mathrm{lm}(Q_0)$ must be the leading monomial of some polynomial P_1 in $R_2 \setminus \{P_0\}$. Then there are three cases for P_1. The first case is that P_1 is a totally new polynomial in $\mathbf{M}$. The second case is that P_1 is in $\mathbf{P}$, and it is not monic since the polynomials in $\mathbf{P}$ are not monic after `AddReduce`. The third case is that a polynomial Q_1 is replaced by P_1 in $\mathbf{M}$. Since the leading monomials of the polynomials in $\mathbf{M}$ before `AddReduce` are different, we have $\mathrm{lm}(P_1) = \mathrm{lm}(Q_0) \neq \mathrm{lm}(Q_1)$. Then $\mathrm{lm}(Q_1)$ must be the leading monomial of some polynomial P_2 in $R_2 \setminus \{P_0, Q_1\}$, and one of the above three cases will happen again. Obvious, the third case can happen finite times. Hence, we can obtain a sequence $Q_0, P_1, Q_1, P_2, Q_2, \ldots, P_k$, such that Q_i is replaced by P_i in $\mathbf{M}$, $\mathrm{lm}(Q_i) = \mathrm{lm}(P_{i+1})$, and P_k is either non-monic or a totally new polynomial in $\mathbf{M}$.
   (a) If P_k is a non-monic polynomial, then after Gaussian elimination, we have $\mathbf{d}_0 + \mathbf{d}_c + \mathbf{d}_{c_1} + \mathbf{d}_{c_2} + \cdots + \mathbf{d}_{c_{k-1}} \leq \mathrm{tdeg}(P_k) + 1 + \mathrm{tdeg}(P_1) + \mathrm{tdeg}(P_2) + \cdots + \mathrm{tdeg}(P_{k-1}) = D_0$, where $c_i = \mathrm{cls}(P_i)$. Moreover, before `AddReduce` we have $\mathbf{d}_0 + \mathbf{d}_c + \mathbf{d}_{c_1} + \mathbf{d}_{c_2} + \cdots + \mathbf{d}_{c_{k-1}} \geq 2 + \mathrm{tdeg}(Q_0) + \mathrm{tdeg}(Q_1) + \mathrm{tdeg}(Q_2) + \cdots + \mathrm{tdeg}(Q_{k-1}) = D_1$. Since $\mathrm{lm}(Q_i) = \mathrm{lm}(P_{i+1})$, we have $\mathrm{tdeg}(Q_i) = \mathrm{tdeg}(P_{i+1})$, then $D_1 - D_0 = 1$. Note that, the new linear polynomials generated by Gaussian elimination have classes bigger than c, hence P_k will not be changed after substitutions, which implies that $Sum(\mathcal{T})$ will decrease.
   (b) If P_k is a totally new polynomial in $\mathbf{M}$, then $\mathbf{d}_0 + \mathbf{d}_c + \mathbf{d}_{c_1} + \mathbf{d}_{c_2} + \cdots + \mathbf{d}_{c_{k-1}} + \mathbf{d}_{c_k} \leq d + 1 + \mathrm{tdeg}(P_1) + \mathrm{tdeg}(P_2) + \cdots + \mathrm{tdeg}(P_{k-1}) + \mathrm{tdeg}(P_k) = D_0$ after Gaussian elimination. Before `AddReduce`, we have $\mathbf{d}_0 + \mathbf{d}_c + \mathbf{d}_{c_1} + \mathbf{d}_{c_2} + \cdots + \mathbf{d}_{c_{k-1}} + \mathbf{d}_{c_k} \geq 2 + \mathrm{tdeg}(Q_0) + \mathrm{tdeg}(Q_1) + \mathrm{tdeg}(Q_2) + \cdots + \mathrm{tdeg}(Q_{k-1}) + d = D_1$. Similarly as a), we have $D_1 - D_0 = 1$. Moreover, after the following substitutions, we still have $\mathbf{d}_0 \leq d$, thus $Sum(\mathcal{T})$ will decrease.

Now we consider `Simplify`. As proof of Proposition 16, we focus on the linear polynomial with lowest class in the first iteration of Loop 3. Suppose this linear polynomial is $L = x_c + l$, where $c = \mathrm{cls}(L)$. Obviously, if $\mathbf{d}_c = d$ before `Simplify`, we can prove that $Sum(\mathcal{T})$ decreases. In the following, consider the case that $\mathbf{d}_c < d$. Suppose there is a polynomial P with class c in $\mathbf{M}$ before `Simplify`. If $P + L$ is a non-monic polynomial, it will not be changed by the following substitutions. Hence, $\mathbf{d}_0 \leq \mathrm{tdeg}(P), \mathbf{d}_c = 1$ after `Simplify`, and $\mathbf{d}_0 = 1, \mathbf{d}_c = \mathrm{tdeg}(P)$ before `Simplify`, which means $Sum(\mathcal{T})$ will not increase. If $P + L$ is a monic polynomial, then it will be in $\mathbf{P}$ after the following substitution, and we will execute `AddReduce` in the following steps of **Triset**. Let T_0 be the value of $Sum(\mathcal{T})$ before `Simplify`, T_1 be the value of $Sum(\mathcal{T})$ after `Simplify` and the following substitutions, and T_2 be the value of $Sum(\mathcal{T})$ after `AddReduce` and the following substitutions. We will prove $T_2 \leq T_0$.

- Suppose we obtain a linear polynomial in the Gaussian elimination after `AddReduce`. In this case, the above analysis about `AddReduce` shows that $T_2 < T'$, where $T' = 2 + d_1 + d_2 + \cdots + d_n$ and d_i is the value of $\mathbf{d}_i$ before `AddReduce`. Obviously, T_1 is equal to $k + d_1 + d_2 + \cdots + d_n$, where $k \geq 2$ is the value of $\mathbf{d}_0$ after `Simplify` and the following substitutions. Hence, $T_1 \geq T' > T_2$. Moreover, if $k \leq \text{tdeg}(P)$, similarly as the case that $P + L$ is non-monic, we can prove that $T_0 \geq T_1$. Since $\text{tdeg}(P) \geq 2$, we have $T_0 \geq T'$. Consequently, $T_0 \geq T' > T_2$.
- Suppose we don't obtain linear polynomials in the Gaussian elimination after `AddReduce`. Then as case 2) in the above analysis about `AddReduce`, $\text{lm}(P + L)$ will be the leading monomial of a polynomial Q_0 in **P** or **M** after `AddReduce` and the following Gaussian elimination. Then, there are three cases: Q_0 is a non-monic polynomial; Q_0 is a totally new polynomial in **M**; a polynomial P_1 is replaced by Q_0 in **M**. Similarly as case 2) in the above analysis about `AddReduce`, we can prove that $T_2 \leq T_0$.

In summary, we proved that $Sum(\mathcal{T})$ will decrease strictly after zero decomposition, and will not increase after `AddReduce` and `Simplify`. It means that the depth of the zero decomposition tree is bounded by $(d-1)n$.

Now we consider the complexity of solving one branch for $\mathbf{BCS}_g$. Obviously, the difference of **BCS** and $\mathbf{BCS}_g$ is the process of Gaussian elimination. We show that compared to the complexity of other operations in **Triset**, the complexity of Gaussian elimination is ignorable. We know that `Simplify` can be executed at most n times, and zero decomposition can be executed at most $(d-1)n$ times. Moreover, `AddReduce` can happen after zero decomposition or `Simplify`, hence can be executed at most dn times. Thus, Gaussian elimination can be executed at most $2dn$ times. The complexity of Gaussian elimination for n vectors with dimension $\sum_{i=2}^{d} \binom{n}{d}$ is bounded by $O(n^{d+2})$. Hence for $\mathbf{BCS}_g$ with $\texttt{Choose}_1$ or $\texttt{Choose}_2$, the complexity of solving one branch is bounded by $O(d(m^2 + (d-1)nm)\log(n)n^{d+3}) + O(2dn^{d+3}) = O((dm^2 + d^2nm)\log(n)n^{d+3})$. Then, we have the following theorem.

Theorem 19. *Let* $\mathbf{P} = \{f_1, f_2, \ldots, f_m\} \subset \mathbb{F}_2[x_1, x_2, \ldots, x_n]$ *with* $\text{tdeg}(\mathbf{P}) = d$ *be the input of* $\mathbf{BCS}_g$*, and set the choose function to be* $\texttt{Choose}_1$ *or* $\texttt{Choose}_2$*. Then the complexity of* $\mathbf{BCS}_g$ *is bounded by* $O((dm^2 + d^2nm)\log(n)n^{d+3}2^{(d-1)n})$.

Remark 3. The probability of $Sum(\mathcal{T})$ increasing after `Simplify` is very low, and we didn't observe this case happened when solving the polynomial systems in our experiments. It means that when consider experimental complexity, executing Gaussian elimination for **M** is redundant, and **BCS** is more efficient than $\mathbf{BCS}_g$. Hence, in the experiments showed in the next section, we implemented **BCS** and compared it with other methods.

5.4. Complexity comparison

We compare the complexity of **BCS** with those of the exist CS algorithms. To the best of the authors' knowledge, the unique result for the complexity of the whole process of a CS algorithm is the complexity bound of $\mathbf{TDCS}_2$, which is $O(2^{\log_2(m)n})$ (Gao and Huang, 2012). It is easy to see that:

1) when $2 \leq d < \log_2(m)$, **BCS** is better than $\mathbf{TDCS}_2$;
2) when $d \geq \log_2(m)$, **BCS** is worse than $\mathbf{TDCS}_2$.

Now we compare the complexity of **BCS** with those of other kinds of algorithms for different degree ranges.

1) $d = 2$: As mentioned before, without any side conditions, the complexity bound of **BCS** in the worst case is $O(n(m+n)2^n)$, and this bound is worse than $4\log_2(n)2^n$, which is the complexity bound of the fast exhaustive search method proposed in (Bouillaguet et al., 2010). For general systems, it is not clear that whether the complexity bound of **BCS** can be improved. However, for other algorithms, its asymptotic complexity can be lower than $O(2^n)$. In this case, the best existing result is $O(2^{0.841n})$ when $m = n$ (Bardet et al., 2013).

2) $d = 3$: To the best of authors' knowledge, it seems that there are few results about the complexity of solving Boolean polynomial systems when $d > 2$. Only some results about the complexity of Gröbner basis algorithms are presented in (Bardet et al., 2003; Bardet, 2004; Bardet et al., 2005). In (Bardet et al., 2003; Bardet, 2004), the authors show that for solving semi-regular systems with $m = n$, the degree of regularity D_{reg} is equal to $0.15n + 1.35n^{1/3} - 1.42 + O(\frac{1}{n^{1/3}})$, hence the complexity of the F5 algorithm is $O(\binom{n}{D_{reg}}^{\omega})$, where $2 < \omega < 3$ is the linear algebra constant. This value is about $O(2^{0.61\omega n})$, and is smaller than $2^{(3-1)n}$, which is the exponential part of the complexity of **BCS**. Therefore, the asymptotic complexity of the F5 algorithm is better than that of **BCS** for semi-regular systems. When considering the input systems without side conditions, the only bound we can know about D_{reg} is $D_{reg} \leq n + 1$. In this case, the complexity of the F5 algorithms is $O(2^{\omega n})$, hence **BCS** is sightly better.
3) $d \geq 4$: For the semi-regular systems with $m = n$, when $4 \leq d \leq 7$, the values of D_{reg} are presented in (Bardet et al., 2003; Bardet, 2004):
   - $d = 4$, $D_{reg} = 0.20n + 1.60n^{1/3} - 1.27 + O(\frac{1}{n^{1/3}})$;
   - $d = 5$, $D_{reg} = 0.24n + 1.79n^{1/3} - 1.11 + O(\frac{1}{n^{1/3}})$;
   - $d = 6$, $D_{reg} = 0.26n + 1.95n^{1/3} - 0.94 + O(\frac{1}{n^{1/3}})$;
   - $d = 7$, $D_{reg} = 0.28n + 2.09n^{1/3} - 0.78 + O(\frac{1}{n^{1/3}})$.

   We can check that in these cases, $O(\binom{n}{D_{reg}}^{\omega})$ is much less than $O(2^{(d-1)n})$, and $\log(\binom{n}{D_{reg}}^{\omega})$ increases much slower than $(d-1)n$ when d increases. For example, when $d = 7$, $\log(\binom{n}{D_{reg}}^{\omega})$ is about $0.86\omega n$, which is much less than $(7-1)n = 6n$. Hence the asymptotic complexity of the F5 algorithm is much better than that of **BCS** for semi-regular systems. Moreover, when considering the input systems without side conditions, similarly as the case $d = 3$, we also have $D_{reg} \leq n + 1$, hence the complexity of the F5 algorithm is bounded by $O(2^{\omega n}) < O(2^{3n})$. In comparison, the exponential part of the complexity of **BCS** is $O(2^{(d-1)n}) \geq O(2^{3n})$, thus **BCS** is worse than F5 algorithm.

In all, the above comparison implies that when d is small, **BCS** is much more efficient than existing CS algorithms and may be comparable with other algorithms, and this is coherent with our experimental observations.

Actually, in a lot of cases, the depth of the zero decomposition tree can be much smaller than $(d-1)n$:

- For example, in practical computation, after `Simplify` or `AddReduce` we may obtain some new linear polynomials or some lower degree monic polynomials, hence $\mathbf{d}_0, \mathbf{d}_i$ may decrease, then $Sum(\mathcal{T})$ decreases much faster. This always happens when the input systems are sparse or have some algebraic structure. In Section 6, we will show that for sparse polynomial systems even when d is big ($d > 4$), **BCS** is still very efficient.
- Another example is the case that the zero set of the input system is large, which implies the dimensions of monic triangular sets corresponding to the solutions are big. Suppose we obtain a monic triangular set with dimension k from one solving branch. This means that after we deal with this solving branch, k entries of $\mathcal{T}$ will be d, thus $Sum(\mathcal{T}) \geq dk + n + 1 - k$. Therefore, the depth of this branch is not larger than $(n+1)d - dk - (n+1-k) = (n+1-k)(d-1)$, and this value is much less than $n(d-1)$ when k is big.[3]

6. Experimental results

In this section we present some experimental results about **BCS**. We have implemented **BCS** with the C language and the CUDD package (http://vlsi.colorado.edu/~fabio/CUDD) by which the Boolean

[3] This explains why **BCS** is much efficient than other algorithms for solving the *Matrix* problems, which have large number of solutions, showed in the Section 6.

polynomials are stored as zero-suppressed binary decision diagrams (ZDDs) (Minto, 1993; Brickenstein and Dreyer, 2009). In our implementation, the `Choose` function is originated from $\texttt{Choose}_2$. That is for a polynomial $P = Ix_c + U$, we define an index

$$(\text{tdeg}(I), term(I), term(U), \text{cls}(I))$$

where $term(P)$ is the number of monomials in P, then choose the polynomial with the smallest index w.r.t. the lexicographical order. The executable file of our implementation is available at https://github.com/hzy-cas/BCS. Our experiments were done on a Macbook Pro with a Intel i7 2.7 GHz CPU (only one core is used), 8G memory, and Mac OS X. We compared **BCS** with the following four methods by solving the same polynomial system on the same platform:

(1) The modified **MFCS** algorithm introduced in Section 3, and used in the experiments of (Gao and Huang, 2012), which is available at https://github.com/hzy-cas/MFCS. We denoted it by $\textbf{MFCS}_\textbf{1}$.
(2) The Gröbner basis routine over Boolean polynomial ring in Magma V2.20-3 w.r.t. graded reverse lexicographic order, denoted by **BGB**. As mentioned in the handbook of Magma, since V2.15, computing the Gröbner bases of an ideal in the Boolean polynomial ring is available, and this routine exploits the properties of Boolean polynomial ring to accelerate the computation.
(3) The Gröbner basis routine over Boolean polynomial ring in SAGE V8.7, denoted by **Polybori**. This routine is implemented by the **Polybori** library, which is designed for solving the problems of Boolean polynomials and uses ZDDs as its data structure (Brickenstein and Dreyer, 2009). Since **Polybori** have good performance in computing the Gröbner basis w.r.t. the lexicographic order, in the experiments, we recorded two groups of data by using the graded reverse lexicographic order and the lexicographic order respectively.
(4) **Cryptominisat** V5.6.8, a SAT-solver which is very efficient for solving SAT problems converted from Boolean polynomial systems hence is a widely used for solving Boolean PoSSo problems (Soos et al., 2009). To convert Boolean PoSSo problems to SAT problems, we used the ANF to CNF converter in SAGE V8.7, which applied a lot of techniques to efficiently convert an ANF to CNF, and recorded the time cost of converting. Note that in our experiments, we wanted to achieve all the solutions of the input systems, therefore we used the parameter "maxsol" such that the solver can output all the solutions.

In our experiments, we solved several groups of Boolean polynomial systems which are generated from some typical algebraic cryptanalysis and reasoning problems. We introduce these problems specifically:

1) *Present*: a polynomial system originated from the key recovery problem of the block cipher Present with one pair of known plaintext and ciphertext (Bogdanov et al., 2007). Here we consider a reduced version of Present which has only 5 rounds. By setting the 80-bit key as variables $\{x_0, x_1, \ldots, x_{79}\}$ and adding some internal variables used to simplify the structure of the systems, we generate a system with 356 variables, 1785 quadratic polynomials. Then we randomly guessed the value of variables $\{x_0, x_1, \ldots, x_{47}\}$, and obtain the input system of our experiment.
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 problem of the cold boot key recovery problem of Serpent (Albrecht and Cid, 2011; Huang and Lin, 2013). The system has 128 variables which corresponding to the 128-bit initial key, 256 polynomials and degree 3.
3) *MayaSbox*: a polynomial system originated from the problem of recovering a secret 4-bit Sbox, $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$, of the block cipher Maya from its input and output difference (Borghoff et al., 2013; Liu and Jin, 2015). The variables of this system are corresponding to the different bits of 16 bytes output $S(0000), S(0001), \ldots, S(1111)$, hence the system has $16 \times 4 = 64$ variables. There are 304 polynomials in this system. 64 of them are quadratic polynomials representing the input and output differences, and 240 of them are polynomials with degree 4, which represent the bijection property of the Sbox.

4) *Canfil*: a polynomial system originated from the stream cipher based on a linear feedback shift register (LFSR) and a filter function (Canteaut and Filiol, 2000; Faugère and Ars, 2003; Gao and Huang, 2012), and has the form

$$\{f(x_1, x_2, \ldots, x_n),\ f(L(x_0, x_1, \ldots, x_{n-1})), \ldots,\ f(L^{m-1}(x_0, x_1, \ldots, x_{n-1}))\}.$$

- *Canfil2*: $n = 64$, $m = 68$, $L = L_1$, $L_1(x_0, x_1, \ldots, x_{63}) = (x_1, x_2, \ldots, x_{63}, x_{63} + x_{59} + x_{46} + x_{45} + x_{36} + x_{30} + x_{24} + x_{18} + x_{14} + x_{11} + x_1 + x_0)$, $f(x_0, x_1, \ldots, x_{63}) = x_5x_{14} + x_0x_{11} + (x_0x_5 + 1)x_7$.
- *Canfil3*: $n = 64$, $m = 68$, $L = L_1$, $f(x_0, x_1, \ldots, x_{63}) = (x_5x_7x_{11} + x_7 + 1)x_{14} + (x_5 + 1)x_{11} + x_0x_5x_7$.
- *Canfil4*: $n = 64$, $m = 68$, $L = L_1$, $f(x_0, x_1, \ldots, x_{63}) = x_0x_{11}x_{14} + (x_0 + 1)x_5x_7 + x_0$.
- *Canfil5*: $n = 64$, $m = 68$, $L = L_1$, $f(x_0, x_1, \ldots, x_{63}) = x_5x_7x_{11}x_{14} + x_5x_7 + x_0$.
- *Canfil6*: $n = 64$, $m = 68$, $L = L_1$, $f(x_0, x_1, \ldots, x_{63}) = x_0x_5x_7x_{14} + x_{11} + x_5x_7$.
- *Canfil7*: $n = 64$, $m = 68$, $L = L_1$, $f(x_0, x_1, \ldots, x_{63}) = x_5x_7x_{14} + x_5x_7x_{11} + (x_0x_5 + 1)x_7 + x_5 + x_0$.
- *Canfil8*: $n = 40$, $m = 60$, $L(x_0, x_1, \ldots, x_{39}) = (x_1, x_2, \ldots, x_{39}, x_{37} + x_{34} + x_{21} + x_{11} + x_5 + x_0)$, $f(x_0, x_1, \ldots, x_{39}) = (x_{25} + x_6x_{11})x_{31} + x_{25} + (x_{11} + 1)x_{18} + x_0x_6x_{11} + x_0x_6$.

5) *Biviuma*: a polynomial system originated from the problem of recovering the internal states of stream cipher Bivium-A (Raddum, 2006; Simonetti et al., 2008), which is a reduced version of stream cipher Trivium. We set the 177-bit internal states as variables, and add two internal variables at each clock of the cipher. Then by 400-bit keystream, we generate a polynomial system with 977 variables and 1062 polynomials from 400-bit keystream. 662 of these polynomials are quadratic, and others are linear.
6) *Biviumb*: a polynomial system originated from the problem of recovering the internal states of the stream cipher Bivium-B (Raddum, 2006; Eibach and Völkel, 2010), which is a reduced version of stream cipher Trivium. By setting the 177-bit internal states as variables, and using 160-bit keystream, we can generate a polynomial system with 177 variables. This system contains 12 polynomials with degree 3, 82 polynomials with degree 2, and 66 polynomials with degree 1. Since this system cannot be directly solved by any method in reasonable time, in the experiments, we guessed 33 variables as the strategy proposed in (Huang and Lin, 2011) to simplify the system.
7) *Matrix*: a polynomial system originated from the Boolean matrix multiplication problem proposed by Stephen Cook in his invited talk at SAT 2004 (Biere, 2008; Cook, 2004; Cook and Nguyen, 2010; Gao and Huang, 2012). The problem is that given two $k \times k$ Boolean matrices A and B, prove $BA = I$ from $AB = I$ by reasoning. By setting the entries of A and B to be $2k^2$ distinct variables, we can obtain k^2 quadratic polynomials from $AB = I$. Then the reasoning problem is equivalent to computing the Gröbner basis or the zero decomposition of these polynomials, then checking whether the polynomials generated by $BA = I$ can be reduced to 0 by the Gröbner basis or by every triangular set in the zero decomposition. In the following tables, *Matrix3*, *Matrix4*, *Matrix5*, *Matrix6* are the polynomial systems corresponding to the problems with order $3, 4, 5, 6$ respectively. Note that, since the number of solutions for $AB = I$ is very huge, evidently a SAT-solver cannot output so many solutions in reasonable time. Therefore, for a SAT-solver, a better way to prove $BA = I$ from $AB = I$ is checking whether the corresponding negative proposition is true. For this purpose, we generate the polynomial system *Matrix-neg* introduced below.
8) *Matrix-neg*: A polynomial system corresponding to the negative proposition of the above matrix multiplication problem. Precisely, we generated a polynomial system consists of polynomials corresponding to $AB = I$, and one polynomial corresponding to $(BA)_{11} = 0$. Here $(BA)_{11}$ is the entry in the first row and first column of BA. It is obvious that this polynomial system has no solution. When the orders of A and B are k, this polynomial system has $2k^2$ variables and $k^2 + 1$ quadratic polynomials. In the following tables, *Matrix3-neg*, *Matrix4-neg*, *Matrix5-neg*, *Matrix6-neg* are the polynomial systems corresponding to the problems with order $3, 4, 5, 6$ respectively.

Besides these polynomial systems generated from cryptanalysis and reasoning, we randomly generated some sparse and dense polynomial systems with different n and different degrees, then solved them in our experiments. Here we set $m = n$, and in this case, we found most of these random generated polynomial systems have 0-3 solutions.

- We generate a sparse polynomial system with degree d by the following way. For each polynomial, we set the number of its monomials with degree d_0 to be $n/2$, where $d_0 = 2, \ldots, d$, and randomly choose each monomial. Then we randomly generated the constant term of this polynomial. At last, we will obtain a sparse inhomogeneous polynomial with n variables, $nd/2$ non-constant terms and total degree d. In the following tables, we denote such polynomial systems with n variables and degree d by $RandSparse(n, d)$.
- To generate a dense polynomial with degree d, for each monomial with degree not bigger than d, we randomly generate the number 0 or 1 with probability 1/2, and if we get 1, then we let this monomial be in the polynomial. By this way, the expectation number of the monomials in this polynomial will be $\frac{1}{2}\sum_{i=0}^{d}\binom{n}{i}$. In the following tables, we denote such polynomial systems with n variables and degree d by $RandDense(n, d)$.

Specific instances of these polynomial systems can be found in the *benchmarks* directory of the implementation of **BCS** at https://github.com/hzy-cas/BCS. In the following table, the time costs of solving these systems by different methods are presented. Note that the input polynomial systems of *Maxtrix* and *Matrix-neg* problems are fixed, while other polynomial systems can be generated by random parameters. Hence in our experiments, except *Maxtrix* and *Matrix-neg* problems, for each other problem, we generated 10 different instances, and the timings presented in these tables are the average time of solving ten instances.

In Table 1, we show the basic parameters of the polynomial systems generated from cryptanalysis and reasoning. Here, n is the number of variables and m is the number of polynomials. In the column "degree", since some systems have polynomials with different degrees, for each degree we wrote down the number of polynomials in the brackets. Table 2, 3, 4 present the timings, which are all given in seconds. In these tables, "#" means crashed, and "$*$" means running over 2 hours without output. Note that, as in (Bard et al., 2007) the timings of **Cryptominisat** in these tables are the sums of the time of converting and the time of solving. Moreover, since for solving these random generated systems, $\mathbf{MFCS}_1$ is always worse than **BCS**, and **Polybori** is always worse than **BGB**, we only list the timings of **BCS**, **BGB** and **Cryptominisat** in Table 3 and Table 4, in order to show the evolution of the timings for these three kinds of methods with different n and d.

From Table 2, we can observe that **BCS** is the most efficient algorithm for solving any of these systems generated from cryptanalysis and reasoning. We think that one reason of **BCS** being so efficient

Table 1
The basic parameters of the input polynomial systems.

Benchmarks	n	m	Degree
MayaSbox	64	304	2(64), 4(240)
Serpent	128	256	3(256)
BiviumB	177	193	1(99), 2(82), 3(12)
Present	356	1833	1(48), 2(1785)
BiviumA	977	1062	1(400), 2(662)
Canfil2	64	68	3(68)
Canfil3	64	68	3(68)
Canfil4	64	68	3(68)
Canfil5	64	68	4(68)
Canfil6	64	68	4(68)
Canfil7	64	68	3(68)
Canfil8	40	60	3(60)
Matrix3	18	9	2(9)
Matrix3-neg	18	10	2(10)
Matrix4	32	16	2(16)
Matrix4-neg	32	17	2(17)
Matrix5	50	25	2(25)
Matrix5-neg	50	26	2(26)
Matrix6	72	36	2(36)
Matrix6-neg	72	37	2(37)

Table 2
Timings for solving polynomial systems from cryptanalysis and reasoning.

Benchmarks	**BCS**	**MFCS$_1$**	**BGB** [grevlex]	**Polybori** [lex]	**Polybori** [grevlex]	**Cryptominisat**
MayaSbox	0.10	0.24	0.90	95.57	93.27	35.19
Serpent	0.40	0.47	110.79	176.00	171.80	292.19
BiviumB	14.17	18.45	61.93	∗	358.98	5512.55
Present	5.97	14.86	511.49	63.00	95.00	21.70
BiviumA	13.52	37.36	2106.29	∗	#	76.27
Canfil2	15.36	16.88	∗	#	#	150.07
Canfil3	41.83	50.13	891.63	#	#	2269.83
Canfil4	1.01	1.95	107.75	15.50	13.90	149.63
Canfil5	28.99	34.05	356.25	277.77	215.32	2778.22
Canfil6	6.69	10.25	208.93	22.7	22.02	2062.02
Canfil7	6.07	6.84	2721.54	#	#	159.55
Canfil8	392.10	∗	∗	#	#	690.64
Matrix3	0.002	0.002	0.66	3.21	1.23	0.53
Matrix3-neg	0.002	0.002	0.02	0.57	0.52	0.55
Matrix4	0.02	0.03	2436.04	#	#	41.46
Matrix4-neg	0.03	0.03	733.38	465.51	450.48	0.48
Matrix5	0.60	14.29	∗	#	#	∗
Matrix5-neg	2.57	10.25	1645.76	#	#	148.40
Matrix6	85.44	∗	∗	#	#	∗
Matrix6-neg	418.88	2716.53	∗	#	#	∗

Table 3
Timings for solving random sparse polynomial systems with $m = n$.

Benchmarks (n, d)	**BCS**	**BGB**	**Cryptominisat**
RandSparse(22, 2)	0.81	9.66	3.37
RandSparse(22, 3)	3.80	#	10.42
RandSparse(22, 4)	11.72	#	20.35
RandSparse(22, 5)	21.69	#	31.46
RandSparse(22, 6)	35.48	#	55.24
RandSparse(22, 7)	43.25	#	58.70
RandSparse(26, 2)	9.51	202.63	19.65
RandSparse(26, 3)	68.94	#	117.98
RandSparse(26, 4)	162.09	#	218.79
RandSparse(26, 5)	356.63	#	389.17
RandSparse(26, 6)	552.36	#	599.02
RandSparse(26, 7)	723.15	#	840.91
RandSparse(30, 2)	112.91	∗	206.68
RandSparse(30, 3)	1198.79	#	3837.57
RandSparse(30, 4)	3185.50	#	∗
RandSparse(30, 5)	5356.85	#	∗
RandSparse(30, 6)	∗	#	∗
RandSparse(30, 7)	∗	#	∗
RandSparse(34, 2)	1147.51	∗	∗
RandSparse(34, 3)	∗	#	∗
RandSparse(34, 4)	∗	#	∗

is that these polynomial systems have some block triangular structure, which means the classes of the polynomials can be divided into different sets. Moreover, most of the polynomials in these systems are sparse. Therefore, the decreasing of $Sum(T)$ for **BCS** is very fast.

From Table 3 and Table 4, we have the following observations.

- For random sparse systems, we can see that **BCS** is the most efficient algorithm. Note that, the influence of the degree of the input systems to the timings is much weaker than we expected.

Table 4
Timings for solving random dense polynomial systems with $m = n$.

Benchmarks (n, d)	**BCS**	**BGB**	**Cryptominisat**
RandDense(18, 2)	1.32	0.44	14.33
RandDense(18, 3)	64.77	41.99	84.75
RandDense(18, 4)	757.13	883.42	363.68
RandDense(18, 5)	*	*	1160.61
RandDense(18, 6)	*	*	*
RandDense(20, 2)	4.95	1.03	53.59
RandDense(20, 3)	429.93	2080.61	420.13
RandDense(20, 4)	7162.80	*	2458.26
RandDense(20, 5)	*	*	*
RandDense(22, 2)	21.55	9.77	193.26
RandDense(22, 3)	2755.51	#	3208.67
RandDense(22, 4)	*	#	*
RandDense(24, 2)	94.23	41.38	1296.9
RandDense(24, 3)	*	#	*
RandDense(24, 4)	*	#	*
RandDense(26, 2)	379.24	249.76	*
RandDense(26, 3)	*	#	*
RandDense(28, 2)	1587.26	*	*
RandDense(28, 3)	*	#	*
RandDense(30, 2)	7038.65	*	*
RandDense(30, 3)	*	#	*

For $d > 3$, when d increases by 1, the timing of **BCS** increases less than double. Moreover, when n increases by 4, the timing increases about $2^{0.9\times 4}$ times. All these observations show that **BCS** can well use the sparse property, hence have a much lower practical complexity. This raises a question: can we achieve a lower asymptotic complexity bound about **BCS** when the input polynomial systems are sparse?

- For random dense systems, when $d = 2$, **BCS** is comparable with other algorithms, and **BGB** is the most efficient one. Moreover, when n increases by 2, the timing of **BCS** increases about 2^2 times, and this is consistent with our asymptotic complexity for quadratic systems. When $d = 3$, **BCS** is comparable with **Cryptominisat**, and **BGB** didn't work well for $n \geq 20$, because the degree of regularity is too high. When $d \geq 4$, **Cryptominisat** becomes the most efficient algorithm, and this is coherent with our prediction. Since for random dense polynomial systems with high degree, few algebraic properties can be exploited, the methods based on the idea of searching the values of variables will be more efficient. Actually, fast exhaustive search algorithms, for example, libFES (Bouillaguet et al., 2010), are much more efficient than all these algorithms for solving random dense polynomial systems, but they are not capable for solving most of the problems in Table 2, since n is too big for these problems. Note that, for these systems, when d increases by 1, the timing of **BCS** increases about 2^6 to 2^7 times, which is much less than 2^n times. It seems that our asymptotic complexity bound can be improved even for random dense systems.

Now, we show why **BCS** is more efficient than other CS algorithms. Compared to $\mathbf{MFCS}_1$ and **MFCS**, the mainly advantage of **BCS** is that the number of branches is smaller. This can be well explained by analyzing the change of $Sum(T)$ after zero decomposition.

- In **MFCS**, $Sum(T)$ decreases slowly. For example, suppose there are k polynomial $\{f_1, f_2, \ldots, f_k\}$ with the highest class and highest degree, and $\mathrm{tdeg}(\mathrm{init}(f_1)) = \mathrm{tdeg}(\mathrm{init}(f_2)) = \cdots = \mathrm{tdeg}(\mathrm{init}(f_k))$. Suppose we have executed zero decomposition w.r.t. f_1. Then, $Sum(T)$ will not change after the next $k-1$ times of decomposition were finished. The reason is that we will choose $f_2, \ldots, f_k$ to do zero decomposition, and $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_n$ will not change. As mentioned in Section 5, in the

worst case, the depth of the zero decomposition tree of **MFCS** can reach the bound proposed in Proposition 13.

- In **MFCS$_1$**, $Sum(\mathcal{T})$ decrease faster than in **MFCS**. For example, if P is the polynomial with shortest initial, then after choosing P to do zero decomposition, in a lot of cases, $\mathrm{init}(P)$ will also be the polynomial with shortest initial, hence $\mathbf{d}_0$ will decrease. However, there are still some cases that $Sum(\mathcal{T})$ doesn't decrease after zero decomposition. For example, if $\mathrm{init}(P)$ is monic, and $\mathbf{d}_c < \mathrm{tdeg}(\mathrm{init}(P)), c = \mathrm{cls}(P)$, since we don't compute add-remainder and `AddReduce` will be executed until all polynomials are monic, then in the following process, we may choose a polynomial Q with $\mathrm{tdeg}(Q) \geq \mathrm{tdeg}(P)$ to do zero decomposition. It means that $\mathbf{d}_c$ will not change and $\mathbf{d}_0$ will not decrease, hence $Sum(\mathcal{T})$ will not decrease.
- In **BCS**, from Section 5, we know that if the choose function is $\texttt{Choose}_1$ (or $\texttt{Choose}_2$), $Sum(\mathcal{T})$(or $Sum(\mathcal{T}')$) strictly decreases after zero-decomposition, which means we don't have "useless" decomposition in **BCS**, hence the number of branches is smallest.

7. Conclusion

In this paper, we present an improved characteristic set algorithm **BCS** to solve Boolean polynomial systems. This algorithm is based on the idea of eliminating variables by addition and some important techniques. We introduce the idea of the zero decomposition tree, by which we convert the problem of estimating the complexity of **BCS** into estimating the complexity of solving one branch and the depth of the tree. We define an index vector about the lowest degree of the non-monic polynomials and monic polynomials with different classes, and give some bounds about the depth of the zero decomposition tree by analyzing the variation of $Sum(\mathcal{T})$, which is the sum of the entries of this index vector. In this way, we obtain some bit-size complexity bounds of **BCS**, which are lower than those of previous characteristic set algorithms. Moreover, by $Sum(\mathcal{T})$, we illustrate how the techniques we used in **BCS** effect the depth of the zero decomposition tree. Furthermore, we test **BCS** by solving some random generated polynomial systems and some polynomial systems generated from cryptanalysis and reasoning problems. Experimental results show that **BCS** is more efficient than the previous characteristic set algorithms, and comparable with other efficient algorithms. It is our future work to see whether we can obtain some lower complexity bounds about the algorithm when the input systems are sparse.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Albrecht, M.R., Cid, C., 2011. Cold boot key recovery by solving polynomial systems with noise. In: ACNS, pp. 57–72.
Aubry, P., Lazard, D., Maza, M.M., 1999. On the theory of triangular sets. J. Symb. Comput. 25, 105–124.
Bard, G.V., Courtois, N.T., Jefferson, C., 2007. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF (2) via SAT-solvers. IACR Cryptology ePrint Archive 24.
Bardet, M., Faugere, J.C., Salvy, B., 2003. Complexity of Gröbner Basis Computation for Semi-Regular Overdetermined Sequences over F2 with Solutions in F2. INRIA report RR-5049.
Bardet, M., Faugère, J.C., Salvy, B., 2004. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In: Proc. ICPSS'04 (International Conference on Polynomial System Solving), pp. 71–75.
Bardet, M., 2004. Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie. Phd Thesis. Université Paris VI, Decomber.
Bardet, M., Faugère, J.-C., Salvy, B., Yang, B.-Y., 2005. Asymptotic behaviour of the degree of regularity of semi-regular quadratic polynomial systems. In: Eighth International Symposium on Effective Methods in Algebraic Geometry. MEGA 2005.
Bardet, M., Faugère, J.C., Salvy, B., Spaenlehauer, P.J., 2013. On the complexity of solving quadratic boolean systems. J. Complex. 29 (1), 53–75.
Bettale, L., Faugere, J.C., Perret, L., 2009. Hybrid approach for solving multivariate systems over finite fields. J. Math. Cryptol. 3 (3), 177–197.
Biere, A., 2008. Linear algebra, Boolean rings and resolution. In: ACA'08. July, Austria.

Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C., 2007. PRESENT: an ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2007. Springer, Berlin, Heidelberg, pp. 450–466.
Borghoff, J., Knudsen, L.R., Stolpe, M., 2009. Bivium as a mixed-integer linear programming problem. In: IMA International Conference on Cryptography and Coding. Springer, Berlin, Heidelberg, pp. 133–152.
Borghoff, J., Knudsen, L.R., Leander, G., Tomsen, S.S., 2013. Slender-set differential cryptanalysis. J. Cryptol. 26 (1), 11–38.
Bouillaguet, C., Chen, H.-C., Cheng, C.-M., Chou, T., Niederhagen, R., Shamir, A., Yang, B.-Y., 2010. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In: CHES 2010. In: LNCS, vol. 6225. Springer, Heidelberg, pp. 203–218.
Boulier, F., Lazard, D., Ollivier, F., Petitiot, M., 1995. Representation for the radical of a finitely generated differential ideal. In: Proc. of ISSAC'95. ACM Press, New York, pp. 158–166.
Bouziane, D., Kandri Rody, A., Maârouf, H., 2001. Unmixed-dimensional decomposition of a finitely generated perfect differential ideal. J. Symb. Comput. 31, 631–649.
Brickenstein, M., Dreyer, A., 2009. PolyBoRi: a framework for Gröbner-basis computations with Boolean polynomials. J. Symb. Comput. 44 (9), 1326–1345.
Canteaut, A., Filiol, E., 2000. Ciphertext only reconstruction of stream ciphers based on combination generators. In: Fast Software Encryption. In: LNCS, vol. 1978. Springer, pp. 165–180.
Chai, F., Gao, X.S., Yuan, C., 2008. A characteristic set method for solving Boolean equations and applications in cryptanalysis of stream ciphers. J. Syst. Sci. Complex. 21 (2), 191–208.
Chou, S.C., 1988. Mechanical Geometry Theorem Proving. D. Reidel, Dordrecht.
Chou, S.C., Gao, X.S., 1990. Ritt-Wu's decomposition algorithm and geometry theorem proving. In: Proc. of CADE-10. In: LNAI, vol. 449. Springer, pp. 207–220.
Courtois, N., Klimov, A., Patarin, J., Shamir, A., 2000. Efficient algorithms for solving over-determined systems of multivariate polynomial equations. In: EUROCRYPT 2000. In: LNCS, vol. 1807, pp. 392–407.
Courtois, N., 2002. Higher order correlation attacks, XL algorithm, and cryptanalysis of toyocrypt. In: ICISC. In: LNCS, vol. 2587. Springer, pp. 182–199.
Cook, S., 2004. From satisfiability to proof complexity and bounded arithmetic. In: SAT 2004, Invited Talk, Vancouver, Canada, 10–13 May.
Cook, S., Nguyen, P., 2010. Logical Foundations of Proof Complexity. Cambridge University Press.
Cox, D., Little, J., O'shea, D., 1992. Ideals, Varieties, and Algorithms. Springer, New York.
Dahan, X., Maza, M.M., Schost, E., Wu, W., Xie, Y., 2005. Lifting techniques for triangular decompositions. In: Proc. ISSAC'05. ACM Press, New York, pp. 108–115.
Eibach, T., Pilz, E., Völkel, G., 2008. Attacking bivium using SAT solvers. In: Büning, H.K., Zhao, X. (Eds.), Theory and Applications of Satisfiability Testing (SAT '08). In: Lecture Notes in Computer Science, vol. 4996. Springer-Verlag, pp. 63–76.
Eibach, T., Völkel, G., 2010. Optimising Gröbner bases on bivium. Math. Comput. Sci. 3 (2), 159–172.
Faugère, J.C., 1999. A new efficient algorithm for computing Gröbner bases (F4). J. Pure Appl. Algebra 139 (1–3), 61–88.
Faugère, J.C., 2002. A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In: Proc. ISSAC 2002, pp. 75–83.
Faugère, J.C., Ars, G., 2003. An Algebraic Cryptanalysis of Nonlinear Filter Generators Using Gröbner Bases. TR No. 4739. INRIA.
Faugère, J.C., Joux, A., 2003. Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases. In: Boneh, Dan (Ed.), Advances in Cryptology-CRYPTO 2003. In: LNCS, vol. 2729. Springer, pp. 44–60.
Gallo, G., Mishra, B., 1991. Efficient algorithms and bounds for Wu-Ritt characteristic sets. In: Effective Methods in Algebraic Geometry. Birkhäuser, Boston, pp. 119–142.
Gao, X.S., van der Hoeven, J., Yuan, C., Zhang, G., 2009. Characteristic set method for differential-difference polynomial systems. J. Symb. Comput. 44 (9), 1137–1163.
Gao, X.S., Huang, Z., 2012. Characteristic set algorithms for equation solving in finite fields. J. Symb. Comput. 47 (6), 655–679.
Gerdt, V., Zinin, M., 2008. A pommaret division algorithm for computing Gröbner bases in Boolean rings. In: Proc. ISSAC 2008. ACM Press.
Huang, Z., Lin, D., 2011. Attacking bivium and trivium with the characteristic set method. In: Progress in Cryptology-AFRICACRYPT 2011. In: LNCS, vol. 6737, pp. 77–91.
Huang, Z., 2012. Parametric equation solving and quantifier elimination in finite fields with the characteristic set method. J. Syst. Sci. Complex. 25 (4), 778–791.
Huang, Z., Lin, D., 2013. A new method for solving polynomial systems with noise over $\mathbb{F}_2$ and its applications in cold boot key recovery. In: Selected Areas in Cryptography. Windsor, Canada. In: LNCS, vol. 7707, pp. 16–33.
Huang, Z., Lin, D., 2017. Solving polynomial systems with noise over F2: revisited. Theor. Comput. Sci. 676, 52–68.
Hubert, E., 2000. Factorization-free decomposition algorithms in differential algebra. J. Symb. Comput. 29, 641–662.
Joux, A., Vanessa, V., 2017. A crossbred algorithm for solving Boolean polynomial systems. In: International Conference on Number-Theoretic Methods in Cryptology. Springer, Cham.
Kalkbrener, M., 1993. A generalized euclidean algorithm for computing triangular representations of algebraic varieties. J. Symb. Comput. 15, 143–167.
Kapur, D., Wan, H.K., 1990. Refutational proofs of geometry theorems via characteristic sets. In: Proc. ISSAC'90. ACM Press, New York, pp. 277–284.
Lazard, D., 1991. A new method for solving algebraic systems of positive dimension. Discrete Appl. Math. 33, 147–160.
Li, X., Mou, C., Wang, D., 2010. Decomposing polynomial sets into simple sets over finite fields: the zero-dimensional case. Comput. Math. Appl. 60 (11), 2983–2997.
Lin, D., Liu, Z., 1993. Some results on theorem proving in geometry over finite fields. In: Proc. ISSAC'93. ACM Press, New York, pp. 292–300.

Liu, G.Q., Jin, C.H., 2015. Differential cryptanalysis of PRESENT-like cipher. Des. Codes Cryptogr. 76 (3), 385–408.
Minto, S., 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In: Proc. ACM/IEEE Design Automation. ACM Press, pp. 272–277.
Möller, H.M., 1993. On decomposing systems of polynomial equations with finitely many solutions. Appl. Algebra Eng. Commun. Comput. 4, 217–230.
Raddum, H., 2006. Cryptanalytic Results on TRIVIUM, eSTREAM, ECRYPT Stream Cipher Project. Report 2006/039.
Simonetti, I., Faugère, J.C., Perret, L., 2008. Algebraic attack against trivium. In: First International Conference on Symbolic Computation and Cryptography, SCC 2008. LMIB, Beijing, China, pp. 95–102.
Soos, M., Nohl, K., Castelluccia, C., 2009. Extending SAT solvers to cryptographic problems. In: Kullmann, O. (Ed.), Theory and Applications of Satisfiability Testing - SAT 2009. SAT 2009. In: Lecture Notes in Computer Science, vol. 5584. Springer, Berlin, Heidelberg.
Sun, Y., Huang, Z., Lin, D., Wang, D., 2016. On implementing the symbolic preprocessing function over Boolean polynomial rings in Gröbner basis algorithms using linear algebra. J. Syst. Sci. Complex. 29 (3), 789–804.
Szanto, A., 1999. Computation with Polynomial Systems. Ph.D. Thesis. Cornell University.
Wang, D., 1993. An elimination method for polynomial systems. J. Symb. Comput. 16, 83–114.
Wu, W.T., 1986. Basic principles of mechanical theorem-proving in elementary geometries. J. Autom. Reason. 2, 221–252.
Zhao, J., Song, J., Zhu, M., Li, J., Huang, Z., Li, X., Ren, X., 2018. PBCS: an efficient parallel characteristic set method for solving Boolean polynomial systems. In: Proceedings of the 47th International Conference on Parallel Processing. ACM, 23.