Solving Polynomial Systems with Noise over $\mathbb{F}_2$: Revisited *

Zhenyu Huang and Dongdai Lin

State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, China
{huangzhenyu,ddlin}@iie.ac.cn

Abstract. Solving polynomial systems with noise over $\mathbb{F}_2$ is a fundamental problem in computer science, especially in cryptanalysis. **ISBS** is a new method for solving this problem based on the idea of incrementally solving the noisy polynomial systems and backtracking all the possible noises. It had better performance than other methods in solving the Cold Boot Key recovery problem. In this paper, some further researches on **ISBS** are presented. We proposed a polynomial ordering scheme by which we can accelerate the incremental solving process of **ISBS**. We present some computation complexity bounds of **ISBS**. Two major improvement strategies, artificial noise-bound strategy and two-direction searching strategy, are proposed and theoretically analyzed. Based on these improvements, we propose a variant **ISBS** algorithm, and by the experiments of solving the Cold Boot key recovery problem of Serpent with symmetric noise, we show that our new algorithm is more efficient than the old one.

Keywords. Boolean polynomial system with noise, Max-PoSSo, ISBS method, Cold Boot attack, Characteristic Set method, Serpent.

MSC Codes. 13P15, 13P25, 14G50, 68Q25, 94A60

1 Introduction

Solving polynomial systems with noise over $\mathbb{F}_2$, which is called as the Max-PoSSo problem over $\mathbb{F}_2$, is the problem of finding an optimal solution of a given Boolean polynomial system, which can satisfy the maximum number of polynomials. It is a fundamental problem in several areas of cryptography, such as algebraic attacks, side-channel attacks and the cryptanalysis of LPN/LWE-based schemes. For example, in the Cold Boot attack, which is a kind of side-channel attack, we can recover the initial key of a block cipher from noisy round keys by solving

* This work was in part supported by National 973 Program of China under Grants No. 2013CB834203 and No. 2011CB302400, the National Natural Science Foundation of China under Grant No. 60970152, the "Strategic Priority Research Program" of the Chinese Academy of Sciences under Grant No. XDA06010701 and IIE's Research Project on Cryptography (No. Y3Z001802).

a Max-PoSSo problem [1, 11]. In computation complexity field, this problem is also significant and is known as the maximum equation satisfying problem [10, 20]. In the general case, this problem is NP-hard even when the polynomials are linear.

In this paper, we will discuss Max-PoSSo problems with all the polynomials are nonlinear. Obviously, Max-PoSSo problems over $\mathbb{F}_2$ are analogous to the well-known Max-SAT problems, thus a natural way to solve a Max-PoSSo problem is converting it into a Max-SAT problem and then solve it with a Max-SAT solver. However, this method has a disadvantages that the original algebraic structure is destroyed. In [1], the authors proposed a method to convert Max-PoSSo problems into mixed integer programming (**MIP**) problems, and then solved it with a **MIP** solver **SCIP**. Essentially, these methods are all based on the idea of searching all the possible values of variables, and their differences are the techniques of pruning redundant branches of the search tree.

In [15], we proposed a new method called **ISBS** for solving Max-PoSSo over $\mathbb{F}_2$. The basic idea of **ISBS** is searching the values of polynomials, which is equal to searching all the possible noises. Precisely speaking, given a noisy polynomials system $\{f_1, f_2, \ldots, f_m\}$, we try to solve polynomial systems $\{f_1 + e_2, f_2 + e_2, \ldots, f_m + e_m\}$, where $(e_1, e_2, \ldots, e_m)$ can be equal to $(0, 0, \ldots, 0), (1, 0, \ldots, 0), \ldots, (1, 1, \ldots, 1)$. Then, the solution of a system $\{f_1 + e_2, f_2 + e_2, \ldots, f_m + e_m\}$ with $(e_1, e_2, \ldots, e_m)$ having the smallest Hamming weight is the solution of the Max-PoSSo problem. In the **ISBS** method, we combined the above idea with the ideas of incrementally solving $\{f_1 + e_1, f_2 + e_2, \ldots, f_m + e_m\}$ and searching all possible $(e_1, e_2, \ldots, e_m)$ with backtracking. By this way, we can prune a lot of search branches when searching the values of $(e_1, e_2, \ldots, e_m)$. From the experimental results of [15], we showed that compared with **SCIP**, **ISBS** had better performances on the Cold Boot Key recovery problem of AES and Serpent.

Since **ISBS** is a new method, some problems are open, for example, the complexity estimation problem. Moreover, there is still a lot of room for improving the efficiency of **ISBS**. Therefore, the motivation of this paper is to do some further research on the properties of Max-PoSSo and **ISBS**, and give some improvements to **ISBS**. The main contributions of this paper are as follows.

In cryptanalysis, in most cases the purpose of solving Max-PoSSo is recovering the true solution, which is the solution of an original noiseless polynomial system. Hence, we study the success rate of recovering the true solution by solving Max-PoSSo, and show that when the error rate are fixed, the success rate will increase as the number of input polynomials increases.

For the complexity estimation problem, it is too hard to analyze the exact complexity of **ISBS**. In this paper, we presents some complexity bounds of **ISBS**, and from these bounds we can illuminate how the randomness of the input system affects the complexity of **ISBS**. Moreover, we show an important fact that the complexity of **ISBS** increases linearly (not exponentially) with the number of input polynomials, which contradicts our intuition.

We propose two improvements of **ISBS** for the general cases. The first one is an ordering scheme for the input polynomials when we use the characteristic set

method [9, 15] as the incremental solving tool, by which we can accelerate the incremental solving process of **ISBS**. The second one is the artificial noise-bound strategy, in which we artificially bound the Hamming weight of the noise vectors and gradually increase the bound until we find the optimal solution. By this way, we can control the search space, thus reducing the computation complexity of **ISBS**.

When the input polynomial system satisfies $m = sn$, where m is the number of polynomials, n is the number of variables and $s \geq 2$ is an integer, we propose a strategy called the s-direction strategy, by which we can reduce the negative influence of the number of the input polynomials to the efficiency of **ISBS**. The idea is dividing the input polynomials into s parts and searching the possible noise by beginning with different parts. This idea is similar with that in [16] for decoding random linear codes. Furthermore, we give some theoretical analysis of this strategy and prove that with this strategy **ISBS** will be more efficient.

We implement a variant **ISBS** algorithm by applying the above improvements, and test it by solving some Cold Boot key recovery problems of Serpent. We compare our experimental results with those in [1, 15], and from these experimental results, we show that by our modification, the efficiency of **ISBS** is improved significantly.

The rest of this paper is organized as follows. In Section 2, we introduce the **ISBS** method. In Section 3, we discuss the incremental solving process in **ISBS**. In Section 4, we analyze the success rate of Max-PoSSo. In Section 5, we present some complexity bounds of **ISBS**. In Section 6, we propose an improvements of **ISBS** by using an artificial noise-bound. In Section 7, the strategy for solving the problem with $m = sn$ is proposed. In Section 8, we show some experimental results. In Section 9, we compare the algorithms for solving Max-PoSSo from the viewpoint of complexity. In Section 10, the conclusions are presented.

2 Solving Max-PoSSo Problems over $\mathbb{F}_2$

Let $\mathbb{F}_2$ be the finite field with two elements and $\mathbb{P} = \{f_1, \ldots, f_m\} \subset \mathbb{F}_2[x_1, \ldots, x_n]$ be a Boolean polynomial system. The polynomial system solving (PoSSo) problem over $\mathbb{F}_2$ is finding a solution $(x_1, ..., x_n) \in \mathbb{F}_2^n$ such that $\forall f_i \in \mathbb{P}$, we have $f_i(x_1, ..., x_n) = 0$. The Max-PoSSo problem over $\mathbb{F}_2$ is defined as below.

Max-PoSSo: *Let $\mathbb{P} = \{f_1, \ldots, f_m\} \subset \mathbb{F}_2[x_1, \ldots, x_n]$ be a Boolean polynomial system. Find a point $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ such that $\{i \in N | f_i(x_1, \ldots, x_n) = 0, f_i \in \mathbb{P}\}$ has maximal cardinality.*

The name "Max-PoSSo" was first proposed in [1]. In the computational complexity field, this problem is sometimes called the maximum equation satisfying problem [10, 20]. Obviously, Max-PoSSo is at least as hard as PoSSo. Moreover, whether the polynomials in $\mathbb{P}$ are linear or not, Max-PoSSo is anf NP-hard problem. Besides Max-PoSSo, in [1], the authors introduced another variant problem: Partial Weighted Max-PoSSo, in which the optimal solution is constrained by another polynomial system and it has to maximize a cost function.

In this paper we only focus on the Max-PoSSo problem, since it is a more fundamental problem and study this problem can help us illustrate the major properties of these similar problems. Moreover, the skills for solving Max-PoSSo can be simply applied into solving Partial Weighted Max-PoSSo and other similar problems.

In the following paragraphs of this paper, unless otherwise stated, the problems we discuss are all over $\mathbb{F}_2$, and we use n to denote the number of variables and m to denote the number of input polynomials. Moreover, we assume $m \geq n$.

2.1 The Incremental Solving and Backtracking Search (ISBS) Method

In this part, we will introduce the **ISBS** method for solving Max-PoSSo problems[15]. As we know, almost all existing algorithms for solving Max-PoSSo problems are based on the idea of searching the values of variables, such as the Max-SAT solvers. The idea of **ISBS** is based on another point of view which is searching the values of polynomials.

Let's show this idea more specifically. Given a noisy polynomial set $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$, for every vector $E = (e_1, e_2, \ldots, e_m) \in \mathbb{F}_2^n$, we can solve the polynomial system $\{f_1 + e_1, f_2 + e_2, \ldots, f_m + e_m\}$ by some method. Hence, we can exhaustively searching all such E in order of increasing Hamming weight and solve the corresponding polynomial system for each E. If the corresponding polynomial system of some E has a solution, then it is the solution of the Max-PoSSo problem.

The above approach uses the most common way in searching E, and obviously there are a lot of redundant computations. For example, if $\{f_1 + e'_1, f_2 + e'_2, \ldots, f_k + e'_k\}$ has no solution for some fixed $(e'_1, e'_2, \ldots, e'_k)$, we don't need to solve any system with the form $\{f_1 + e'_1, f_2 + e'_2, \ldots, f_k + e'_k, f_{k+1} + e_{k+1}, \ldots, f_m + e_m\}$, since it will always be a contradiction system. Therefore, in order to avoid this kind of redundant computations, **ISBS** combines the incremental solving method and the backtracking search method with the above idea.

First let's introduce the incremental solving method. Given a polynomial set $\mathbb{P}$, we can solve it by some algebraic method, such as the Characteristic Set(CS) method [4, 9] and the Gröbner Basis method [7, 8]. Note that the SAT-solver is not suitable for the incremental solving method, since it cannot represent a lot of solutions with a simple form. We denote the output results of such a solving algorithm with input $\mathbb{P}$ as $\texttt{Result}(\mathbb{P})$. From $\texttt{Result}(\mathbb{P})$, all the solutions of $\mathbb{P}$ can be derived easily. We remind the reader that, for different methods, $\texttt{Result}(\mathbb{P})$ can be different.

For example, if we use the CS method to solve $\mathbb{P}$, $\texttt{Result}(\mathbb{P}) = \cup_i \mathcal{A}_i$ is a group of triangular sets(A triangular set $\mathcal{A}_i$ is a polynomial set which can be easily solved, and its precise definition will be given in next section). If we use the Gröbner Basis method to solve $\mathbb{P}$, $\texttt{Result}(\mathbb{P})$ is the Gröbner Basis of idea $< \mathbb{P} >$. In the following paragraphs, for any solving method, when the polynomial system $\mathbb{P}$ has no solution, we set $\texttt{Result}(\mathbb{P}) = \{1\}$.

We show that given $\texttt{Result}(\mathbb{P})$ and a polynomial g, $\texttt{Result}(\{\texttt{Result}(\mathbb{P}), g\})$ can be achieved. For example, for the CS method, we need to compute each $\texttt{Result}(\{\mathcal{A}_i, g\})$ and output the union of them. For the Gröbner Basis method, we need to compute the Gröbner Basis of the idea generated by the new polynomial set. Therefore, given a polynomial system $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$, $\texttt{Result}(\mathbb{P})$ can be achieved by recursively computing

$$\texttt{Result}(\{f_1\}), \texttt{Result}(\{\texttt{Result}(\{f_1\}), f_2\}), \ldots,$$

and this is the incremental solving method. Now let's present the processes of **ISBS**.

(i) We try to incrementally solve $\{f_1 + e_1, f_2 + e_2, \ldots, f_i + e_i\}$ for i from 1 to m with each $e_i = 0$. If $\texttt{Result}(\{f_1, f_2, \ldots, f_i\}) = 1$ for some i, we flip e_i to 1 and continue solving the remainning polynomials based on $\texttt{Result}(\{f_1, f_2, \ldots, f_i{+}1\})$. At last, we will obtain a candidate $\texttt{Result}(\{f_1{+}e_1, f_2{+}e_2, \ldots, f_m{+}e_m\})$ where $(e_1, \ldots, e_m)$ is equal to some fixed $(e'_1, \ldots, e'_m)$. Then, we set the backtracking index k to be m.

(ii) Then, in order to obtain a better candidate, we search all the possible values of $(e_1, \ldots, e_m)$ with backtracking based on the value $(e'_1, \ldots, e'_m)$. That is for i from k to 1 we find the first e'_i which are equal to 0, and similarly as step (i) we try to incrementally solve $f_{i+1}, \ldots, f_m$ based on $\texttt{Result}(\{f_1 + e'_1, \ldots, f_i + 1\})$. If we find a better candidate $\texttt{Result}(\{f_1 + e'_1, \ldots, f_i + 1, f_{i+1} + e''_{i+1}, \ldots, f_m + e''_m\})$, we set k to be m, replace $(e'_1, \ldots, e'_m)$ with $(e'_1, \ldots, e'_{i-1}, 1, e''_{i+1}, \ldots, e''_m)$, and do step (ii) again. Otherwise, set $k = i{-}1$ and if $k > 0$ do step (ii) again.

(iii) Finally, we have searched all the possible $(e_1, \ldots, e_m)$ and obtain the optimal solution.

In the following algorithm, we present the **ISBS** method specifically. In this algorithm, a vector $(e_1, e_2, \ldots, e_s) \in \mathbb{F}_2^s$ with $s \leq m$ is called a *noise vector*. Moreover, we call the Hamming weight of a noise vector its *noise weight* and a bound of the noise weight a *noise-bound*.

For the proof of the correctness and termination of **ISBS**, the reader is referred to [15].

Algorithm 1: ISBS algorithm

input : A polynomial sets $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$.
output: $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ s.t. $\{i \in N | f_i(x_1, \ldots, x_n) = 0, f_i \in \mathbb{P}\}$ has maximal cardinality.

1 Compute $\texttt{Candidate}(\mathbb{P}, \emptyset, m, 0)$;
2 Let t, u, $\{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}$ and $E = (e_1, e_2, \ldots, e_m)$ be the corresponding output ; /* Here t is always equal to m */
3 $u_{bound} \leftarrow u - 1$, $\mathbb{S} \leftarrow \mathbb{Q}_m$;
4 Let $\mathbb{S}_1$ be the output of $\texttt{Backtracking}(\mathbb{P}, E, \{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}, u_{bound}, u)$;
5 **if** $\mathbb{S}_1 \neq \emptyset$ **then** $\mathbb{S} \leftarrow \mathbb{S}_1$
6 Get $(x_1, \ldots, x_n)$ from $\mathbb{S}$ and **return** $(x_1, \ldots, x_n)$;

Function: `Candidate`

input : $\mathbb{P} = \{f_{s+1}, f_{s+2}, \ldots, f_m\}$: a polynomial set,
`Result`$(\mathbb{R})$: the common zero of a polynomial set $\mathbb{R}$,
u_{bound}: a noise-bound;
u: the noise weight of $(e_1, \ldots, e_s)$;
output: t: a index number,
u: the noise weight of $(e_1, \ldots, e_{s+t})$,
$\mathbb{QS} = \{\mathbb{Q}_s, \mathbb{Q}_{s+1}, \ldots, \mathbb{Q}_{s+t}\}$: a sequence of zero sets,
$E = (e_{s+1}, e_{s+2}, \ldots, e_{s+t})$: a vector

$\mathbb{Q}_s \leftarrow$ `Result`$(\mathbb{R})$;
for i *from* $s+1$ *to* m **do**
  $\mathbb{P}_i \leftarrow \{\mathbb{Q}_{i-1}, f_i\}$, solve $\mathbb{P}_i$ and achieve `Result`$(\mathbb{P}_i)$;
  if `Result`$(\mathbb{P}_i) = \{1\}$ **then** `/* In this case, Result`$(\mathbb{Q}_{i-1}, f_i + 1) =$ `Result`$(\mathbb{Q}_{i-1})$ `*/`
    $\mathbb{Q}_i \leftarrow \mathbb{Q}_{i-1}$, $e_i \leftarrow 1$, $u \leftarrow u + 1$;
    if $u > u_{bound}$ **then**
      $t \leftarrow i - s$, and **break;**
  else if `Result`$(\mathbb{P}_i)$ *and* $\mathbb{Q}_{i-1}$ *have the same zero set* **then** `/* In this case, Result`$(\mathbb{Q}_{i-1}, f_i + 1) = \{1\}$ `*/`
    $\mathbb{Q}_i \leftarrow \mathbb{Q}_{i-1}, e_i \leftarrow 0$;
  else
    $\mathbb{Q}_i \leftarrow$ `Result`$(\mathbb{P}_i)$, $e_i \leftarrow 0$;
return t, u, $\{\mathbb{Q}_s, \mathbb{Q}_{s+1}, \ldots, \mathbb{Q}_{s+t}\}$, $(e_{s+1}, e_{s+2}, \ldots, e_{s+t})$;

Function: `Backtracking`

input : $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$: a polynomial set,
$E = (e_1, e_2, \ldots, e_m)$: a noise vector,
$\{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}$: a sequence of zero sets
u_{bound}: a noise-bound;
u: the noise weight of E;
output: A zero set $\mathbb{S}$.

$k \leftarrow m$, $\mathbb{S} \leftarrow \emptyset$; `/*` k `is the backtracking index */`
while $k \geq 1$ **do**
  if $e_k = 0$, $\mathbb{Q}_k \neq \mathbb{Q}_{k-1}$ *and* $u + 1 \leq u_{bound}$ **then**
    $e_k \leftarrow 1$, $u \leftarrow u + 1$;
    Solve $\mathbb{P}_k = \{\mathbb{Q}_{k-1}, f_k + 1\}$ and achieve `Result`$(\mathbb{P}_k)$;
    $\mathbb{Q}_k \leftarrow$ `Result`$(\mathbb{P}_k)$;
    Set t, u, $\{\mathbb{Q}_k, \mathbb{Q}_{k+1}, \ldots, \mathbb{Q}_{k+t}\}$, $(e_{k+1}, \ldots, e_{k+t})$ to be the output of `Candidate`$(\{f_{k+1}, f_{k+2}, \ldots, f_m\}, \mathbb{Q}_k, u_{bound}, u)$;
    $k \leftarrow k + t$;
    if $u \leq u_{bound}$ **then** `/* In this case,` $k = m$ `*/`
      $\mathbb{S} \leftarrow \mathbb{Q}_m$, $u_{bound} \leftarrow u - 1$;
  else
    $u \leftarrow u - e_k$, $k \leftarrow k - 1$;
return $\mathbb{S}$.

Remark 1. This version of **ISBS** is slightly different with the version in [15]. We add the comparisons of the zero sets of $\texttt{Result}(\mathbb{P}_i)$ and $\mathbb{Q}_{i-1}$ (Step 8) in `Candidate`. When this condition is true, we always have $\texttt{Result}(\mathbb{Q}_{i-1}, f_i + 1) = \{1\}$. Thus, in this case we don't need to check whether $\texttt{Result}(\mathbb{Q}_{i-1}, f_i + 1)$ is equal to $\{1\}$ in `Backtracking` (That is what we did in [15]). Therefore, we add a condition $\mathbb{Q}_k \neq \mathbb{Q}_{k-1}$ in Step 3 of `Backtracking` to verify whether this case happens.

Note that checking whether $\texttt{Result}(\mathbb{P}_i)$ and $\mathbb{Q}_{i-1}$ have the same zero set is very easy, thus the cost is much less than that of solving $\{\mathbb{Q}_{i-1}, f_i + 1\}$. For example, when using the characteristic set method as the incremental solving tool, the number of points in the zero set can be compute easily, thus we only need to check whether $\texttt{Result}(\mathbb{P}_i)$ and $\mathbb{Q}_{i-1}$ have the same number of points.

3 The Incremental Solving Process

From the processes of **ISBS**, it is easy to see that each time we call the function `Candidate`, we are incrementally solving a polynomial system. As we know, solving a polynomial system is a NP problem. In **ISBS**, we need to solve lots of similar polynomial systems. The differences of these systems are the constant terms, thus the time for solving them are almost the same. Obviously, if we cannot solve the input system in a short time, **ISBS** will not end in a practical time. Thus, in this paper, we assume that the input system can be solved in constant time. A lot of polynomial systems satisfy this assumption, for example, the systems generated by presenting the Serpent round keys w.r.t. the initial key, which were considered in [1, 15] and the experiments in Section 8 of this paper.

3.1 The Characteristic Set Method

In our implementation of **ISBS**, we used the Characteristic Set(CS) method as the incremental solving tool. The CS method is an important method in symbolic computation [6, 19], and a powerful tool for solving Boolean polynomial systems [4, 9, 13]. Here we introduce the CS method for solving Boolean polynomial systems. For more details, the reader is referred to [9].

First, let's introduce some basic notations. For a Boolean polynomial $P \in \mathbb{F}_2[x_1, x_2, \ldots, x_n]$, 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 $\mathrm{init}(P)$.

A sequence of nonzero polynomials

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

is a *triangular set* if either $r = 1$ and $A_1 = 1$, or $0 < \mathrm{cls}(A_1) < \cdots < \mathrm{cls}(A_r)$. For example, $\{x_1 + c_1, x_2 + c_2, \ldots, x_n + c_n\}$ is a triangular set which corresponds to the point $(c_0, c_1, \ldots, c_n)$.

A Boolean polynomial P is called *monic*, if $\text{init}(P) = 1$. Moreover, if the elements of a triangular set are all monic, we call it a monic triangular set.

Given a polynomial system $\mathbb{P}$, we use $\text{Zero}(\mathbb{P})$ to denote the common zero of this system, that is

$$\text{Zero}(\mathbb{P}) = \{(a_1, \cdots, a_n), a_i \in \mathbb{F}_2, s.t., \forall f_i \in \mathbb{P}, f_i(a_1, \cdots, a_n) = 0\}.$$

Obviously, for a monic triangular set $\mathcal{A} : A_1, A_2, \ldots, A_r$, $|\text{Zero}(\mathcal{A})| = 2^{n-r}$.

By the characteristic set method, we can decompose $\text{Zero}(\mathbb{P})$ into the disjoint union of the zero sets of several monic triangular sets. That is

$$\text{Zero}(\mathbb{P}) = \bigcup_i \text{Zero}(\mathcal{A}_i)$$

where each $\mathcal{A}_i$ is a monic triangular sets and $\text{Zero}(\mathcal{A}_i) \cap \text{Zero}(\mathcal{A}_j) = \emptyset$ for any $i \neq j$. Obviously, if the number polynomials in $\mathcal{A}_i$ is d_i, we have $|\text{Zero}(\mathbb{P})| = \sum_i 2^{n-d_i}$.[1]

In the following Algorithm 4, we present the **ICS** algorithm which is an incremental solving algorithm based on the CS method. This algorithm can be use in the Step 3 of `Candidate` and Step 5 of `Backtracking`.

Algorithm 4: ICS

input : A polynomial system $\{f_1, f_2, \ldots, f_m\}$
output: $\text{Zero}(f_1, f_2, \ldots, f_m)$

1 $\mathbb{RS} \leftarrow \{\emptyset\}$;
2 **for** *i from 1 to m* **do**
3 $\quad \mathbb{QS} \leftarrow \emptyset$;
4 $\quad$ Let k be the number of elements in $\mathbb{RS} = \{\mathbb{R}_1, \ldots, \mathbb{R}_k\}$; `/* ` $\mathbb{R}_i$ `is a monic triangular set */`
5 $\quad$ **for** *j from 1 to k* **do**
6 $\quad\quad$ For every linear polynomials $L = x_c + l$ in $\mathbb{R}_j$, substitute x_c with l in f_i; `/* The simplification process */`
7 $\quad\quad$ $\mathbb{R}_j = \mathbb{R}_j \cup \{f_i\}$;
8 $\quad\quad$ $\mathbb{QS} = \mathbb{QS} \cup \mathbf{BCS}(\mathbb{R}_j)$;
9 $\quad$ $\mathbb{RS} \leftarrow \mathbb{QS}$;
10 $\quad$ **if** $\mathbb{RS} = \emptyset$ **then return** $\emptyset$
11 **return** $\mathbb{RS}$;

[1] This formula of counting the number of solutions can be used in the checking process mentioned in Remark 1(The step 8 of `Candidate`).

Algorithm 5: BCS

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

$\mathbb{P}^* = \{\mathbb{P}\}, \mathcal{A}^* = \emptyset$;
while $\mathbb{P}^* \neq \emptyset$ **do** `/* P* is a group of polynomial sets */`
  Select a polynomial set $\mathbb{Q}$ from $\mathbb{P}^*$;
  Let $\mathcal{A}$ and $\mathbb{Q}^*$ be the output of `Triset(`$\mathbb{Q}$`)`;
  if $\mathcal{A} \neq \emptyset$ **then** $\mathcal{A}^* = \mathcal{A}^* \cup \{\mathcal{A}\}$ $\mathbb{P}^* \leftarrow \mathbb{P}^* \cup \mathbb{Q}^*$;
return $\mathcal{A}^*$.

Function: `Triset`

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

$\mathbb{P}^* \leftarrow \emptyset$, $\mathcal{A} \leftarrow \emptyset$ and $n_{monic} = 0$;
while $\mathbb{P} \neq \emptyset$ **do**
  Let $\mathcal{A}'$ and $\mathbb{P}'$ be the output of `Simplify(`$\mathbb{P}$`)`;
  if $\mathcal{A}', \mathbb{P}' = \emptyset$; **then return** $\emptyset$ and $\mathbb{P}^*$ **else** $\mathcal{A} = \mathcal{A} \cup \mathcal{A}'$, $\mathbb{P} = \mathbb{P}'$ **if** *The elements in* $\mathbb{P}$ *are all monic* **then**
    if *For every class, there is only one element in* $\mathbb{P}$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \mathbb{P}$;
      return $\mathcal{A}$ and $\mathbb{P}^*$;
    else
      $\mathbb{P}' \leftarrow$`AddReduce(`$\mathbb{P}$`)`;
      if $\mathbb{P}' = \emptyset$ **then return** $\emptyset$ and $\mathbb{P}^*$ **else** $\mathbb{P} \leftarrow \mathbb{P}'$
  else `/* The decomposition process */`
    Choose a polynomial $P = Ix_c + U$ with lowest degree from $\mathbb{P}$;
    $\mathbb{P}_1 \leftarrow (\mathbb{P} \setminus \{P\}) \cup \mathcal{A} \cup \{I, U\}$, and $\mathbb{P}^* \leftarrow \mathbb{P}^* \cup \{\mathbb{P}_1\}$;
    $\mathbb{P} \leftarrow (\mathbb{P} \setminus \{P\}) \cup \{x_c + U\} \cup \{I + 1\}$;
    $n_{monic} \leftarrow n_{monic} + 1$;
return $\mathcal{A}$ and $\mathbb{P}^*$.

Function: `Simplify`

input : A polynomial set $\mathbb{P}$
output: A monic triangular set $\mathcal{A}$, and a polynomial set $\mathbb{P}$

$\mathcal{A} \leftarrow \emptyset$;
if $1 \in \mathbb{P}$ **then return** $\emptyset$ and $\emptyset$ **while** $\mathbb{P}$ *has a linear polynomial* $P = x_c + L$ **do** `/*` $\mathrm{cls}(P) = c$ `*/`
  Substitute x_c with L for the other elements in $\mathbb{P}$;
  $\mathcal{A} = \mathcal{A} \cup \{x_c + L\}$;
  if $1 \in \mathbb{P}$ **then return** $\emptyset$ and $\emptyset$
return $\mathcal{A}$ and $\mathbb{P}$.

Function: `AddReduce`

input : A polynomial set $\mathbb{P}$
output: A polynomial set $\mathbb{P}'$

1 Sort the elements of $\mathbb{P}$ by classes and obtain polynomial sets $\mathbb{Q}_1, \mathbb{Q}_2, \ldots, \mathbb{Q}_t$;
`/* The class of elements in` $\mathbb{Q}_i$ `is` c_i `*/`
2 **for** $i \leftarrow 1$ *to* t **do**
3 Let $Q \in \mathbb{Q}_i$ be the polynomial with lowest total degree;
4 $\mathbb{Q}_i \leftarrow \mathbb{Q}_i \setminus \{Q\}$, $\mathbb{P}' = \mathbb{P}' \cup \{Q\}$;
5 **while** $\mathbb{Q}_i \neq \emptyset$ **do**
6 Choose an element $Q_j \in \mathbb{Q}_i$, $\mathbb{Q}_i \leftarrow \mathbb{Q}_i \setminus \{Q_j\}$;
7 $Q_j \leftarrow Q_j + Q$;
8 **if** $Q_j = 1$ **then return** $\emptyset$ **if** $Q_j \neq 0$ **then** $\mathbb{P}' = \mathbb{P}' \cup \{Q_j\}$
9 **return** $\mathbb{P}'$.

The **BCS** algorithm presented here has some differences with the **MFCS** algorithm in [9]. We add `Simplify` into the algorithm, and in Step 15 we choose the polynomial with lowest degree, while in **MFCS** we choose the polynomial with highest class. These modification can highly decrease the complexity of the CS method.

Now let's show some results about the complexity of **BCS**. For **BCS**, when we say a solving branch, we mean one execution of `Triset`. The complexity of **BCS** can be achieved by combining the complexity of a solving branch and the number of branches.

Similarly as the proof in [9], we can prove that the bit-size complexity of `Triset` is $O(dmn^{d+2}\log(n))$, where d is the degree of these polynomials. Furthermore, we estimate the number of solving branches when the input polynomial system is random. Obviously, all these solving branches consist a binary tree. Note that, we generate a new branch by zero-decomposition of $P = Ix_c + U$ at Step 15-17 of `Triset`. Then, we add $I + 1$ to the current branch and I to the new generated branch. This means that the two branches both have a new polynomial with degree at most $d - 1$. Since we assume that P is a random polynomial, this new polynomial will be non-monic. Thus, when we do the zero-decomposition next time, we will choose this new generated polynomial, or another polynomial with lower degree. In any cases, the degree of this chosen polynomial will be at most $d - 1$. Then after decomposing this polynomial, we we generate a new polynomial with degree at most $d - 2$. Thus, after do the zero-decomposition $d - 1$ times, we will achieve a linear polynomial. Then by `Simplify`, we can use this linear polynomial to eliminate one variable. Since we have n variables, the above process can be repeated at most n times. Hence, we can do the zero-decomposition at most $(d-1)n$ times, which means the number of solving branches is bounded by $2^{(d-1)n}$. Furthermore, we have the bit-size complexity of **BCS** can be bounded by $O(2^{(d-1)n}dmn^{d+2}\log(n))$ when solving a random polynomial system.

Note that in Step 15 of `Triset`, if we choose the polynomial which is generated from the f_i with smallest index i, and has the lowest degree, **BCS**

is equivalent to **ICS**. By this modification, the above complexity bound of **BCS** will not change, thus the bit-size complexity of **ICS** is also bounded by $O(2^{(d-1)n}dmn^{d+2}\log(n))$.

Although the above complexity bound is worse than the complexity of exhaustive search, **BCS** and **ICS** can be much faster than the exhaustive search when solving a lot of practical systems. For these systems, the complexities of **BCS** and **ICS** are still unknown. Unfortunately, the polynomial systems we talk about in this paper are such systems, so we don't know the precise complexity of solving them by **ICS**, but we know some properties about the complexity. This will be discussed in Section 3.2

3.2 Some Properties of Incremental Solving

In this section, we will talk about the following four problems.

1. The comparison of the efficiencies of the incremental solving method and the general solving method.
2. Why we use the incremental solving method.
3. Why we choose the CS method for incremental solving.
4. Properties of the complexity of incremental solving.

Problem 1. It is well known that the complexity of solving a polynomial system highly depends on n, the number of its variables, and m, the number of polynomials. In most cases, a system with large m will be more easily to be solved. For example, when $m = O(n^d)$, where d is the degree of the polynomials, the system can be solved in a polynomial time by the Gröbner Basis method. Note that, by the incremental solving method, at the beginning of the process, m is much smaller than n. It means that the incremental solving method is less efficient than the general solving method in most cases.

However, there are still some cases in which the incremental solving method is more efficient than the general solving method. Here we present an example. We consider the above Serpent system generated form round keys without noise. In this case, the polynomial system has 128 variables, and $128r$ polynomials, where r is the number of rounds. We tried to solve this kind of systems by the CS method and the Gröbner method under general solving mode and incremental solving mode. For the CS method, we used our implementation of algorithms **BCS** and **ICS**. For the Gröbner Basis method, we used the Boolean Gröbner Basis function in Magma (Version 2.20) under graded reverse lexicographical order. Table 1 lists the running time of solving these systems by using different methods. The column "r" is the number of rounds.

Table 1. Solving the Serpent problem by GB and CS

r	GB	Incre. GB	**BCS**	**ICS**
1	0.08 s	0.23 s	0.08 s	0.05 s
2	132.41 s	3.47 s	0.67 s	0.18 s

From the results, we can see that for the Gröbner Basis method, when $r = 2$, the incremental solving method is better than the general solving method. For the CS method, when $r = 1, 2$, the incremental solving method is always better than the general solving method.

From our solving experiments on this system and other systems, we found that when the "hardness" of each input polynomial is different, the incremental solving method may be better than the general solving method. For example, in the above system with $r = 2$, the polynomials generated by the first round keys are easier than the polynomials generated by the second round keys. We remind the reader that the above system is a special instance, and in most cases, the general solving method is better.

Probelm 2. A natural question is why we use the incremental solving method when it is not so efficient in most cases. Actually, as mentioned before, our purpose of using the incremental solving method in **ISBS** is to decrease the redundant computations. Compared to the general solving method, when solving Max-PoSSo, by combining with backtracking, the incremental solving method has two major advantages.

- If the input system doesn't have good randomness, by incremental solving method and backtracking, we can prune a lot of redundant searching branches.
- For different noises, the polynomial systems are almost the same. By incremental solving method, we can save the former results then reduce the repeating computations.

Here, we present a example to show the effect of the incremental solving method. For the above Serpent system with $r = 1$, we try to solve the corresponding Max-PoSSo problem under a noise-bound 4. By the general solving method, we have to solve $\sum_{i=0}^{4} \binom{128}{i} \approx 2^{23.4}$ different systems. Even if we assume that the time for solving one system is about 0.05 seconds (the best result in the above table), it will take $2^{19.07}$ seconds to achieve the optimal solution. In comparison, our experiment show that, by our implementation of **ISBS** method, we can solved this problem in about 2 seconds. Obviously, this is a significantly improvement.

Problem 3. It is easy to see that a method for solving PoSSo problems can be modified into an incremental solving method. To the best of the authors' knowledge, the major methods for solving PoSSo problems are: the exhaustive search method, the SAT solver, the Gröbner Basis method, the XL method, and the Characteristic Set method.

The exhaustive search and the SAT solver are both based on the idea of searching all the possible points in $\mathbb{F}_2^n$. As mentioned before, their disadvantage in incremental solving is we cannot represent and store the former results by an easy way, since at the beginning of solving, a lot of points can satisify the first several polynomials. Then it is hard to use the former results to reduce repeating computations. Thus, these two methods are not suitable for incremental solving.

Since the XL method and the Gröbner Basis method are similar, we only need to compare the Gröbner Basis method with the CS method. The Gröbenr Basis method and the CS method are all algebraic solving method. They can represent and store the former results well. Thus, they are suitable for incremental solving. The difference is their solving efficiencies. From the experimental results in Table 1, we can see that for solving the Serpent problem, **ICS** is much faster than the incremental Gröbner Basis method (IGB).

Actually, from our experiments, we found that for the systems which can be solved by incremental methods in a practical time, **ICS** is faster than IGB. We think the main reason is that the systems which can be solved by incremental solving methods always have some triangular structure, therefore by **ICS** we can make use of this structure feature. In comparison, in the incremental solving process, since m is small at the beginning, IGB cannot well use the advantage of its fast Gaussian elimination techniques. Another reason is that we have our own implementation of the CS method, so **ICS** is implemented by C language. In comparison, we don't have a good implementation of the Gröbner Basis method, so IGB is implemented by Magma functions.

In summary, since the better performance of the CS method in incremental solving, we choose it as the incremental solving approach in our implementation of **ISBS**.

Problem 4. As mentioned before, when a system can be solved easily, we don't know the exact complexity of solving it. What we know is the following property of the complexity.

First, let's define a kind of polynomial systems which are called *perfectly random polynomial systems*. For a ordered polynomial system $\{f_1, f_2, \ldots, f_m\}$ with $m \geq n$, we can define a map $\mathcal{S}_n : \mathbb{F}_2^n \to \mathbb{F}_2^n : \mathcal{S}_n(x) = (f_1(x), f_2(x), \ldots, f_n(x))$.

Definition. A ordered polynomial system with $m \geq n$ is called a *perfectly random system*, if the corresponding $\mathcal{S}_n$ is a bijection.

For a perfectly random system, since $\mathcal{S}_n$ is a bijection, we can know that, for any point $e = (e_1, e_2, \ldots, e_n)$, it has a unique preimage w.r.t. $\mathcal{S}_n$. It implies that for any noise vector $(e_1, e_2, \ldots, e_n)$, $\{f_1(x) + e_1, f_2(x) + e_2, \ldots, f_n(x) + e_n\}$ has a unique solution. Then the complexity of incremental solving the system can be divided into two parts. The first part is the complexity of solving the first n polynomials, and the second part is the complexity of solving the remaining $m - n$ polynomials. We can denote the first part by C_n. Note that, this C_n may be exponential in general cases. Under our assumption that polynomial systems can be easily solved, this C_n may be a polynomial w.r.t to n, m. After solving the first n polynomials, we will obtain a unique solution. If we use **ICS**, this solution will be represented by $\{x_1 + c_1, x_2 + c_2, \ldots, x_n + c_n\}$. Then the process of incrementally solving the rest $m - n$ polynomials is easy. For **ICS**, we only need to substitute each x_i by c_i in each of the remaining $m - n$ polynomials. We suppose the complexity of substitution for one polynomial can be bounded

by C. In general cases, C is equal to $O(\binom{n}{d}d)$, where d is the degree of these polynomials. If the densities of these polynomials (the number of monomials) are almost same, then C is equal to $O(dt)$, where t is the number of monomials. Note that compared to C_n, this C is very small. Then, the complexity of solving these remaining polynomials is about $(m-n)C$.

For most polynomial systems in cryptanalysis, since they have good randomness property, after we solve the first n polynomials, we may have several solutions or no solution. Hence, the behavior of the complexity of incremental solving these systems are similar as above.

3.3 The polynomial ordering problem

From the process of incremental solving , it is easy to see that the order of the input polynomials $\{f_1, f_2, \ldots, f_m\}$ is a major factor that affect the efficiency of the whole solving process. Thus, a critical problem for the incremental solving process is that given a polynomial system, in what order we can solve the system fastest.

Intuitively, a natural ordering scheme is putting a "easy" polynomial in front of a "hard" one, since with the results achieved from the "easy" polynomials we can simplify the "hard" ones. Obviously, the definitions of "easy" and "hard" should be combined with the algebraic solving method. In this section, we will discuss this problem when using **ICS** as the solving tool.

For **ICS**, when we say a component, we mean a loop of Step 5 in the Algorithm. It is obvious that the number of total components significantly affect the efficiency of **ICS**.

As we mentioned before, the main principle for a good polynomial ordering scheme is that we should use the "easy" polynomials achieved from the zero decomposition of the former polynomials to simplify the latter ones. From the description of **ICS**, we can observe that the main simplification process of **ICS** is doing the substitution by the appeared linear polynomials. For a polynomial P, if we solve it with **CS**, then we will achieve a group of triangular sets, we denote the number of these sets by N_{cs} and the number of linear polynomials in these triangular sets by N_{lin}. By these notations, we proposed the following two principles for polynomial ordering.

1. The linear polynomials in $\mathrm{Zero}(f_1, f_2, \ldots, f_{i-1})$ can be used to simplify f_i.
2. For the front polynomial, the corresponding value of $\sum_{i=0}^{N_{cs}} 2^{n-N_{lin}}$ should be as small as possible.

Principle 1 means that the classes of the linear polynomials in $\mathrm{Zero}(f_1, \ldots, f_{i-1})$ should be smaller than $\mathrm{cls}(f_i)$. Principle 2 means that the output of the zero decomposition of the front polynomial should contain more linear polynomials and less components.

We can define a *order vector* $(c, term(I) + term(R), term(I) \cdot term(R))$ for a polynomial $P = Ix_c + R$, where $c = \mathrm{cls}(P)$, $term(P)$ denotes the number of monomials in P. Then we can order the input polynomials by ordering their

corresponding order vector. We order these order vectors by the lexicographical order, and say a polynomial f_1 has lower ordering than a polynomial f_2, which denoted by $f_1 \prec f_2$, if the order vector of f_1 has lower ordering than that of f_2. Based on this ordering scheme, we have:

1. In **ICS**, the classes of the linear polynomials, which are achieved from the zero decomposition of $\text{Zero}(f_1, \ldots, f_{i-1})$, are not bigger than $\text{cls}(f_{i-1})$. Furthermore, from the definition of the order vector, we know that $\text{cls}(f_{i-1}) \leq \text{cls}(f_i)$ if $f_{i-1} \prec f_i$, which means Principle 1 can always be satisfied.
2. The satisfaction of Principle 2 is based on our extensive experiments, by which we found that in most cases $\sum_{i=0}^{N_{cs}} 2^{n-N_{lin}}$ increases with $term(I) + term(R) = term(P)$ decreasing. Moreover, for two polynomials $P_1 = I_1 x_c + R_1$, $P_2 = I_2 x_c + R_2$, with $term(P_1) = term(P_2)$, if the sizes of I_1 and R_1 are more balanced, then the value of $\sum_{i=0}^{N_{cs}} 2^{n-N_{lin}}$ corresponding to P_1 is smaller. The theoretical proof of this observation is still an open problem.

The following table is some experimental results about the running time of **ICS** for solving the same systems with different polynomial orders. These polynomial systems are generated from the key schedule process of Serpent. We present round keys of Serpent by the polynomials with respect to the initial keys, and try to solve the initial keys with the first two round keys. In this case, the polynomial system has 128 variables and 256 equations. The reason we choose this kind of problems is that in the following sections, we will solve the problem of recovering the initial key of Serpent from some noisy round keys, and incrementally solving this kind of systems is a basic process when solving that problem.

In the experiments, we used three different orders and for each order we tried to solve 100 groups of systems. In this table, "Random" means we sort the input polynomials by a random order, and "Natural" means we sort the polynomials by the natural order of the round key bits.

Table 2. Solving the Serpent problem by ICS with different polynomial orders

Order	min t	avg. t	max t
Random	•	•	•
Natural	0.187 s	0.233 s	0.297 s
Proposed	0.156 s	0.180 s	0.249 s

• means time out: 1 hour without output

From the experimental results, we can observe that for this Serpent key recovery problem, the order we proposed is the best one compared to the others, and the efficiency of **ICS** is indeed strongly related to the polynomial order. Actually, for most polynomial systems generated from cryptanalysis, if the order is bad, such as the random order, sometimes we cannot solve the systems by incremental method. Natural order is a good order for most cryptanalysis

problem, since the polynomials we generate will become more complex as the cryptographic algorithm proceeds further, but it is not the best one in most cases.

4 The Success Rate of Recovering the True Solution

In cryptanalysis, the input polynomial system $\mathbb{P}$ of a Max-PoSSo problem is always originated from another system $\mathbb{P}_0$ which has a solution. Moreover, the difference between $\mathbb{P}_0$ and $\mathbb{P}$ is that t polynomials of them have different constant terms. If $|\mathbb{P}_0| = |\mathbb{P}| = m$, the ratio $r = t/m$ is called the error rate of $\mathbb{P}$. We call the solution of $\mathbb{P}_0$ *the true solution* of $\mathbb{P}$.

Actually, in cryptanalysis, the major motivation of solving a Max-PoSSo problem is to recover the true solution which is the sensitive information in most cases. Unfortunately, sometimes when the error rate is big, the optimal solution of a Max-PoSSo problem is not the true solution. Therefore, in order to predict the effectiveness of recovering the true solution via the Max-PoSSo model, we have to estimate the probability of recovering the true solution based on the basic information, such as the number of variables n, the number of polynomials m and the error rate r, of the input polynomial system. Moreover, we want to know how this probability changes when m, n and r change. In this section, we will present some results about this problem under the assumption that the input system is random and balance. In the following of this section, when we say the *success rate*, we mean the probability of recovering the true solution by solving a Max-PoSSo problem.

Given a input system, $\mathbb{P} = \{f_1(x_1, x_2, \ldots, x_n), \ldots, f_m(x_1, x_2, \ldots, x_n)\}$ with $m \geq n$, we define a map $\mathcal{S}_m : \mathbb{F}_2^n \to \mathbb{F}_2^m$, with $\mathcal{S}_m(x) = (f_1(x), f_2(x), \ldots, f_m(x))$. Assume for any $x_1 \neq x_2$, $\mathcal{S}_m(x_1)$ and $\mathcal{S}_m(x_2)$ are independent, and $\mathrm{Im}(\mathcal{S}_m)$ is uniform distributed over $\mathbb{F}_2^m$. Obviously, this assumption is reasonable when $f_1, f_2, \ldots, f_n$ are random and balance. Under this assumption, we have the following proposition about the success rate.

Proposition 1 *Let $\mathbb{P}$ be the input system of a Max-PoSSo problem with error rate r. Suppose $\mathbb{P}$ is a random balance polynomial system which has n variables and m polynomials. Then the success rate $\mathcal{P}$ is equal to* $(1 - \frac{\sum_{i=0}^{rm}\binom{m}{i}-1}{2^m})^{2^n}$.

Proof: Suppose $\mathbb{P}$ is generated from $\mathbb{P}_0$. Then t polynomials of $\mathbb{P}$ and $\mathbb{P}_0$ have different constant terms. We denote the differences between the elements of $\mathbb{P}$ and $\mathbb{P}_0$ by $e'_1, e'_2, \ldots, e'_m$, then $e'_i = 0$ or 1 for $1 \leq i \leq m$. The optimal solution of $\mathbb{P}$ is the true solution when the following condition is satisfied:

- For any noise vector $(e_1, e_2, \ldots, e_m)$ such that its noise weight $w \leq t = mr$ and $(e_1, e_2, \ldots, e_m) \neq (e'_1, e'_2, \ldots, e'_m)$, $\texttt{Result}(f_1 + e_1, \ldots, f_m + e_m) = \{1\}$.

$\texttt{Result}(f_1 + e_1, \ldots, f_m + e_m) = \{1\}$ implies that for any point $x_0 \in \mathbb{F}_2^n$, $(f_1(x_0), f_2(x_0) \ldots, f_m(x_0)) \neq (e_1, e_2, \ldots, e_m)$. Therefore, the number of noise vectors satisfying the above condition is $\sum_{i=0}^{t}\binom{m}{i} - 1$. By the assumption that

$\mathrm{Im}(\mathcal{S}_m)$ is uniform distributed over $\mathbb{F}_2^m$, for any point x_0, the probability of $\mathcal{S}_m(x_0)$ being not equal to these noise vectors is $1 - \frac{\sum_{i=0}^{t}\binom{m}{i}-1}{2^m}$. Since there are 2^n points in $\mathbb{F}_2^n$, then $\mathcal{P}$ is equal to the probability of the above condition being satisfied, which is $(1 - \frac{\sum_{i=0}^{mr}\binom{m}{i}-1}{2^m})^{2^n}$. □

To observe the change of $\mathcal{P}$ when m increases, we need to following proposition.

Proposition 2 *Let $r \leq 1/2$ be a fixed real number and $m_1 r$ be a positive integer. For any integers m_2, such that $m_2 > m_1$ and $(m_2 - m_1)r$ is an integer, we have*

$$\frac{\sum_{i=0}^{m_2 r}\binom{m_2}{i}}{2^{m_2}} \leq \frac{\sum_{i=0}^{m_1 r}\binom{m_1}{i}}{2^{m_1}}.$$

Moreover, the equality holds when $r = 1/2$.

Proof: The completely strict proof of this proposition is complicated. Here we present an approximate proof by the normal distribution theory, and a strict proof of this proposition when $1/r$ is an integer is given in Appendix.

By the central limit theorem, we know that when m is large enough, $\frac{\sum_{i=0}^{mr}\binom{m}{i}}{2^m}$ is approximately equal to

$$\frac{1}{\sqrt{m\pi/2}}\int_{-\infty}^{mr} e^{-\frac{1}{2}(\frac{t-m/2}{\sqrt{m/4}})^2}dt = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{mr-m/2}{\sqrt{m/4}}} e^{-\frac{1}{2}s^2}ds$$

Thus, it is sufficient to prove $\frac{m_1 r - m_1/2}{\sqrt{m_1/4}} \geq \frac{m_2 r - m_2/2}{\sqrt{m_2/4}} \Leftrightarrow (\sqrt{m_1} - \sqrt{m_2})(2r - 1) \geq 0$. Obviously, when $r \leq 1/2$, this inequality holds, and when $r = 1/2$, the equality holds. □

From the above proposition, we can deduce that when n, r are fixed, $\frac{\sum_{i=0}^{mr}\binom{m}{i}-1}{2^m}$ decreases with m increasing, which means $\mathcal{P}$ increase with m increasing. Similarly, we can prove that when n, m are fixed, $\mathcal{P}$ decreases with r increasing. Actually, in cryptanalysis, r and n are always fixed in most problems, such as the Cold Boot key recovery problem we will introduced in Section 8, and the probabilistic algebraic attacks on LFSR-based stream ciphers [2]. It means that by increasing the number of input polynomials, we can increase the success rate of recovering the true solution.

5 The Complexity Bound of ISBS

In this section, we will analyze the size of the backtracking search tree, and then give some bounds of the complexity of **ISBS** under the following complexity assumption about the incremental solving process.

According to Problem 4 of Section 3.2, we assume that the polynomial system we consider in the following of this paper has the property that, the behavior of

the complexity of incremental solving the system is similar as that of incremental solving a perfectly random system. Then, we suppose the complexity of incrementally solving a system $\{f_1+e_1, f_2+e_2, \ldots, f_k+e_k\}$ is C_k for any $1 \leq k \leq n$, and for any $k > n$ the complexity of solving $\{f_1+e_1, f_2+e_2, \ldots, f_k+e_k\}$ based on $\texttt{Result}(f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1})$ is C. Here we set $C_0 = 0$. For these C_k and C, we have $C_{k_1} > C_{k_2}$ for $k_1 > k_2$ and C is much less than C_n.

Our complexity bounds of **ISBS** will be represented by these C_k and C. In general cases, we can set C_k to be the complexity bound of solving a polynomial system with i polynomials by **ICS** proposed in Section 3.1, and C to be the complexity of substitution proposed in Problem 4 of Section 3.2. As we mentioned before, when the assumption that the polynomial systems can be easier solved, is valid, the exact values of these C_k are unknown. Hence, we cannot get the exact value of these complexity bounds of **ISBS**. However, these bounds are still meaningful. Actually, these bounds can help us to observe the relation between the backtracking search process and the incremental solving process, and inspire us to improve the algorithm.

Now let's introduce the backtracking search tree of **ISBS**. This binary search can be generated by following the procedures of the algorithm.

First, let the root node of the tree be the empty set. We set the depth of the root node to be 0, and use a pointer $\mathcal{M}$ pointing to the root node. Then run the algorithm and generate the new nodes by the following operations.

- In `Candidate`, each time we set the value of some e_i, we generate a new node $f_i + e_i$, and draw an edge from the node pointed by $\mathcal{M}$ to this new node, then let $\mathcal{M}$ point to this new node.
- In `Backtracking`, each time we decreasing the backtracking index k by 1, we let $\mathcal{M}$ point to the parent node of the node pointed by $\mathcal{M}$ currently. Besides, when we set the value of some e_i, we generate a new node $f_i + e_i$, and draw a edge from the node pointed by $\mathcal{M}$ to this new node, then let $\mathcal{M}$ point to this new node.

This way, we can generate a binary tree with depth m after running **ISBS**. It is easy to see that in this tree the external (or called root-to-leaf) paths with length k can be one-to-one corresponding to the processes of incrementally solving a system with k polynomials in **ISBS**, and we call this tree *the search tree* of **ISBS**. Now, we can use the size of the search tree to estimate the complexity of the whole algorithm.

Remark 2. For a path $\{f_1+e_1, f_2+e_2, \ldots, f_s+1, \ldots, f_k+e_k\}$ in the search tree, the corresponding solving process in **ISBS** may be that we incrementally solve $\{f_1 + e_1, f_2 + e_2, \ldots, f_s\}$ first, and then solve $\{f_1 + e_1, f_2 + e_2, \ldots, f_{s-1}, f_{s+1} + e_{s+1}, \ldots, f_k + e_k\}$ based on $\texttt{Result}(f_1 + e_1, f_2 + e_2, \ldots, f_{s-1})$. We assume the complexity of the above solving process is equal to the complexity of incrementally solving $\{f_1+e_1, f_2+e_2, \ldots, f_s+1, \ldots, f_k+e_k\}$, since the number and the structure of polynomials involved in the solving process are the same.

Since the depth of the binary tree is m, the number of paths in the whole search tree is roughly bounded by 2^m. However, our search tree is not a perfect binary tree, since a lot of subtrees will be pruned in the following four cases:

(1) `Result`$(\mathbb{P}_i) = \{1\}$ in Step 4 of `Candidate`
(2) `Result`$(\mathbb{P}_i)$ and $\mathbb{Q}_{i-1}$ have the same zero set in Step 8 of `Candidate`.
(3) $u > u_{bound}$ in Step 6 of `Candidate`
(4) $u + 1 > u_{bound}$ in Step 3 of `Backtracking`

From the four cases, we can conclude two major factors which affect the size of the search tree. Cases (1) and (2) are corresponding to the first factor, the randomness of the input system. Cases (3) and (4) are corresponding to the second factor, the value of the noise-bound. In this section, we will mainly talk about the first factor. Now we present some propositions about the external paths of the search tree

Proposition 3 *The leaf node with depth $k < m$ has the form $f_k + 1$*

Proof: Suppose the leaf node $f_k + e_k$ is the terminal node of the external path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$. From the processes of **ISBS**, we can check that in the above cases (1), (2) and (4), although we prune some paths, we will not generate a terminal node. Only in case (3), we can generate a terminal node $f_k + e_k$ with $k < m$. In this case, we have `Result`$(\{\mathbb{Q}_{k-1}, f_k\}) = \{1\}$ and the noise weight of $(e_1, e_2, \ldots, e_{k-1}, 1)$ exceeds the noise-bound. Then, the terminal node we generate is $f_k + 1$. □

Proposition 4 *Suppose the input of* **ISBS** *is a polynomials system $\{f_1, \ldots, f_m\}$. For the search tree of* **ISBS**, *we denote the number of paths with length k by $\mathcal{N}_k$, for $1 \leq k \leq m$. We define a map $\mathcal{S}_k : \mathbb{F}_2^n \to \mathbb{F}_2^k$, with $\mathcal{S}_k(x) = (f_1(x), f_2(x), \ldots, f_k(x))$. Then, we have*

(a) $\mathcal{N}_k \leq |Im(\mathcal{S}_k)| \leq 2^t$, for any $1 \leq k < m$, where $t = min(k, n)$.
(b) $\mathcal{N}_m \leq |Im(\mathcal{S}_{m-1})| \leq 2^t$, where $t = min(m-1, n)$.

Proof: a) For a path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$, we have `Result`$(\{f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k\}) \neq \{1\}$, otherwise, this path will be pruned at some node $f_s + e_s$, where $1 \leq s < k$. Thus, $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$, which means we can build an injection from paths with length $k < m$ to $\mathrm{Im}(\mathcal{S}_k)$. Hence, $\mathcal{N}_k \leq |\mathrm{Im}(S_k)| \leq 2^t$.

b) Suppose the first $m-1$ nodes of a path with length m is $f_1 + e_1, f_2 + e_2, \ldots, f_{m-1} + e_{m-1}$. Obviously, `Result`$(\{f_1 + e_1, f_2 + e_2, \ldots, f_{m-1} + e_{m-1}\} \neq \{1\}$.

- If `Result`$(f_1 + e_1, f_2 + e_2, \ldots, f_{m-1} + e_{m-1}, f_m) \neq \{1\}$, there is an external path whose terminal node is f_m in the tree. Since the weight of $(e_1, e_2, \ldots, e_{m-1}, 0)$ is smaller than $(e_1, e_2, \ldots, e_{m-1}, 1)$, the new leaf node $f_m + 1$ will not be generated.
- If `Result`$(f_1 + e_1, f_2 + e_2, \ldots, f_{m-1} + e_{m-1}, f_m) = \{1\}$, we set $e_m = 1$ in `Candidate`, which means we only generate a node $f_m + 1$, and the leaf node f_m will not be generated.

Hence, in both cases, there is only one path with length m whose first $m-1$ nodes are fixed. As the proof of (a), we can build an injection from all the external paths with length m to $\mathrm{Im}(\mathcal{S}_{m-1})$. This implies that $P_m \leq |\mathrm{Im}(\mathcal{S}_{m-1})| \leq 2^t$, where $t = \min(m-1, n)$. □

When we estimate the size of the search tree, the influence of the noise-bound is hard to predict. Like in the proof of Proposition 4, the only thing we can predict is that by using the noise-bound at depth m, the node $f_{m-1} + e_{m-1}$ of an external path with length m always has unique child node. Thus, in order to estimate the complexity bound of **ISBS**, we need to build a tree in which the noise-bound only used to prune the redundant nodes with depth m. To do this, we need to slightly modify **ISBS** and build the corresponding search tree. That is we only compare the noise weight with u_{bound} when $i = m$ in `Candidate` and $k = m$ in `Backtracking`. Obviously, the complexity of the modification algorithm is higher than that of **ISBS**. If we can obtain a complexity bound of this algorithm, it is also a complexity bound of **ISBS**. We call the search tree of this modification algorithm the *quasi search tree*. By analyzing the structure of this tree, we can estimate the complexity bound we needed and illuminate the influence of the randomness of the input system to the complexity of **ISBS** more clearly. We have the following proposition about a quasi search tree.

Proposition 5 *Let $\mathcal{N}_k$ and $\mathcal{S}_k$ be defined as Proposition 4. For a quasi search tree, we have:*

(a) $\mathcal{N}_k = |Im(\mathcal{S}_k)|$, for any $1 \leq k < m$.
(b) $\mathcal{N}_m = |Im(\mathcal{S}_{m-1})|$
(c) All the external paths have length m.

Proof: (a) Similarly as the proof of Proposition 4, we can build an injection $\mathcal{M}$ from paths with length k to $\mathcal{S}_k$. It is sufficient to show that $\mathcal{M}$ is also a surjection. Since we only use the noise-bound at depth m, for any $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(S_k)$, path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$ will not be pruned in the tree. Thus, $\mathcal{M}$ is a surjection.

(b) Similarly as (a), it is sufficient to show the injection we build is also a surjection. For any point $(e_1, e_2, \ldots, e_{m-1}) \in \mathcal{S}_{m-1}$, we have $\mathtt{Result}(f_1+e_1, f_2+e_2, \ldots, f_{m-1}+e_{m-1}) \neq \{1\}$. Since we only use the noise-bound at depth m, path $f_1 + e_1, f_2 + e_2, \ldots, f_{m-1} + e_{m-1}$ will exist in the tree. If $\mathtt{Result}(f_1 + e_1, f_2 + e_2, \ldots, f_m) \neq \{1\}$, the external path $f_1+e_1, f_2+e_2, \ldots, f_m$ will exist in the tree. Otherwise, the external path $f_1 + e_1, f_2 + e_2, \ldots, f_m + 1$ will exist in the tree. Hence, each $(e_1, e_2, \ldots, e_{m-1}) \in \mathcal{S}_{m-1}$ has a preimage.

(c) As we mentioned in Proposition 3, in a search tree a external path with length $k < m$ occurs when the noise weight exceeds the noise-bound. However, in a quasi search tree, the noise-bound is only used at depth m, thus no external path with length $k < m$ will exist. □

Theorem 6 *Let the input system of* **ISBS** *be* $\{f_1, f_2, \ldots, f_m\}$. *Suppose* C_i *and* C *are defined as in the beginning of this section. Denote the complexity of* **ISBS** *by* C_{ISBS}. *We have the following two complexity bounds*

(i) $C_{ISBS} \leq \sum_{k=1}^{n} |Im(\mathcal{S}_k)|(C_k - C_{k-1}) + \sum_{k=n+1}^{m-1} |Im(\mathcal{S}_k)|C + |Im(\mathcal{S}_{m-1})|C$

(ii) $C_{ISBS} \leq 2^n C_n + (m-n)2^n C$

Proof: (i) For a path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$, if we have already taken into account the complexity of solving all paths with length $k-1$, then the cost of solving $\texttt{Result}(\{f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k\})$ is $C_k - C_{k-1}$. From Proposition 5, we have $\mathcal{N}_k = \mathrm{Im}(\mathcal{S}_k)$, thus, the complexity of solving all the paths with length k based on the former results is $\mathrm{Im}(\mathcal{S}_k)(C_k - C_{k-1})$. Hence, by adding up the complexities of solving paths with different lengths, we have the conclusion.

(ii) From (c) of Proposition 5, we have the number of external paths $\mathcal{N} = \mathcal{N}_m \leq 2^n$. Since any external path has length m, the complexity of such path is $C_n + (m-n)C$. Therefore, $C_{ISBS} \leq 2^n(C_n + (m-n)C)$. □

From Theorem 6, we have the following three important facts about the complexity of **ISBS**.

Fact 1. From the proof of conclusion (ii) of Theorem 6, we can conclude that *for any input polynomial system, the number of external paths in the search tree of* **ISBS** *is always bounded by* 2^n.

Fact 2. Note that, if we consider the complexity of **ISBS** corresponding to the quasi search tree, the equality of (i) will hold. By this equality, we can better illuminate the influence of the randomness of the input system to the complexity.

For a random input system, we can assume that $\mathcal{S}_n$ is an injection, then for any $k > n$ we have $\mathcal{S}_k$ is also an injection. Moreover, for any $k < n$ we have $\mathcal{S}_k$ is a surjection and the number of preimage for any points in $\mathbb{F}_2^k$ is the same. Then $|\mathrm{Im}(\mathcal{S}_k)| = 2^k$ for $k \leq n$ and $|\mathrm{Im}(\mathcal{S}_k)| = 2^n$ for $k > n$, which means that each $\mathrm{Im}(\mathcal{S}_k)$ has the maximal cardinality. It is clear that $C_{i+1} - C_i$, $C > 0$, thus *the complexity of* **ISBS** *is maximal for a random input system.*

When the randomness of the input system is not good, some points in $\mathbb{F}_2^n$ may have the same image, which means $|\mathrm{Im}(\mathcal{S}_k)| < 2^k$. Moreover, $|\mathrm{Im}(\mathcal{S}_k)|$ will decrease with the randomness of the system becoming worse, and the complexity of **ISBS** will be smaller.

Fact 3. Since C_n and C are not relevant to m, conclusion (ii) of Theorem 6 shows that:

When the number of variables n *is fixed, the complexity of* **ISBS** *increases linearly (not exponentially) with the number of the input polynomials* m.

As we mentioned before, for a general system, the influence of the noise-bound to the complexity is hard to predict. However, its influence is significant,

thus in the next section we will propose a method to control the noise-bound artificially, by which we can greatly improve the efficiency of **ISBS**.

6 The Artificial Noise-bound Strategy

In **ISBS**, u_{bound} is determined by the current candidate solution. If the noise weight of the optimal solution is small, the procedure of u_{bound} decreasing to this value will take a long time. In this case, we can construct an artificial bound and then try to find the optimal solution under this bound. If there isn't a solution, we can gradually increase the artificial bound by a step size s until we find the optimal solution. Actually, the idea of this method is similar to the idea of exhaustively searching all the possible noise with incremental noise weight. The specific steps of this method are showed in Algorithm 2.

Algorithm 2: ISBS$_b$ algorithm

input : A polynomial sets $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$;
A step size s.
output: $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ s.t. $\{i \in N | f_i(x_1, \ldots, x_n) = 0, f_i \in \mathbb{P}\}$ has maximal cardinality.

1 Compute `Candidate`$(\mathbb{P}, \emptyset, m, 0)$;
2 Let t, u_m, $\{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}$ and $E = (e_1, e_2, \ldots, e_m)$ be the corresponding output ; `/* Here` t `is always equal to` m `*/`
3 $u_{bound} \leftarrow 0$, $\mathbb{S} \leftarrow \mathbb{Q}_m$;
4 **while** $u_{bound} < u_m - 1$ **do**
5 $\quad u_{bound} = \min(u_{bound} + s, u_m - 1)$;
6 $\quad$ Let $\mathbb{S}_1$ be the output of `Backtracking`$(\mathbb{P}, E, \{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}, u_{bound}, u_m)$;
7 $\quad$ **if** $\mathbb{S}_1 \neq \emptyset$ **then** $\mathbb{S} \leftarrow \mathbb{S}_1$ and **break**
8 Get $(x_1, \ldots, x_n)$ from $\mathbb{S}$ and **return** $(x_1, \ldots, x_n)$;

Similarly as the method proposed in last section, we can give some complexity bounds of **ISBS**$_b$ by analyze the structure of the corresponding search tree. Let $\mathcal{N}_k$ and $\mathcal{S}_k$ be defined as in the last section, we have the following proposition about the search tree.

Proposition 7 *Let $Im(\mathcal{S}_k^0)$ be the points $(e_1, e_2, \ldots, e_k)$ in $Im(\mathcal{S}_k)$ with $e_k = 0$. If we assume that the elements in $Im(\mathcal{S}_k)$ are uniformly distributed over $\mathbb{F}_2^k$, then under a noise-bound u we have*

(1) $\mathcal{N}_k = |Im(\mathcal{S}_k)| \cdot \sum_{i=0}^{u} \binom{k}{i}/2^k + (|Im(\mathcal{S}_{k-1})| - |Im(\mathcal{S}_k^0)|)\binom{k-1}{u}/2^{k-1}$, *for* $k < m$.
(2) $\mathcal{N}_m = |Im(\mathcal{S}_{m-1})| \cdot \sum_{i=0}^{u} \binom{m-1}{i}/2^{m-1}$.

Without the assumption of uniformly distribution, we have

(3) $\mathcal{N}_k \leq min(2^k, 2^n, \sum_{i=0}^{u} \binom{k}{u})$
(4) $\mathcal{N}_m \leq min(2^{m-1}, 2^n, \sum_{i=0}^{u} \binom{m-1}{u})$

Proof: (1) There are two kinds of paths with length k, the external paths and the internal paths. For any external path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$, we have $e_k = 1$, $\texttt{Result}(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1}+e_{k-1}) \neq \{1\}$ and $\texttt{Result}(\{f_1+e_1, f_2+e_2, \ldots, f_{k-1} + e_{k-1}, f_k\}) = \{1\}$, which means $(e_1, e_2, \ldots, e_{k-1}) \in \mathrm{Im}(\mathcal{S}_{k-1})$ and $(e_1, e_2, \ldots, e_{k-1}, 0) \notin \mathrm{Im}(\mathcal{S}_k)$. Let

$$M = \{(a_1, a_2, \ldots, a_{k-1}, 0) \in \mathbb{F}_2^k | (a_1, a_2, \ldots, a_{k-1} \in \mathrm{Im}(\mathcal{S}_{k-1})\},$$

then $(e_1, e_2, \ldots, e_{k-1}, 0) \in M \setminus \mathrm{Im}(\mathcal{S}_k^0)$. Moreover, we have the noise weight of $(e_1, e_2, \ldots, e_{k-1}, 1)$ is $u+1$, which means the noise weight of $(e_1, e_2, \ldots, e_{k-1}, 0)$ is u. By the assumption of uniformly distribution, in $M \setminus \mathrm{Im}(\mathcal{S}_k^0)$, the number of vectors whose noise weight is u is $(|\mathrm{Im}(\mathcal{S}_{k-1})| - |\mathrm{Im}(\mathcal{S}_k^0)|)\binom{k-1}{u}/2^{k-1}$.

For any internal path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$, we have $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$. Since it is not pruned by the noise-bound u, we have the noise weight of $(e_1, e_2, \ldots, e_k)$ is not greater than u. Thus, the number of such vectors is $|\mathrm{Im}(\mathcal{S}_k)| \cdot \sum_{i=0}^{u} \binom{k}{i}/2^k$.

Since the above two kinds of vectors are one-to-one corresponding to external paths and internal paths, by adding the numbers of these vectors, we obtain the conclusion.

(2) All the path with length m are external paths. Note that in last section, we use the noise-bound at depth m for the quasi search tree, thus, same as the proof of (b) of Proposition 5, we can prove that $\mathcal{N}_m$ is equal to the number of the internal paths with length $m-1$, which is $|\mathrm{Im}(\mathcal{S}_{m-1})| \cdot \sum_{i=0}^{u} \binom{m-1}{i}/2^{m-1}$.

(3) For a path $f_1 + e_1, f_2 + e_2, \ldots, f_k + e_k$ with $k < m$, no matter whether it is an internal path or an external path, we have $(e_1, e_2, \ldots, e_k) \in \mathrm{Im}(\mathcal{S}_k)$. Thus $\mathcal{N}_k \leq |\mathrm{Im}(\mathcal{S}_k)| \leq \min(2^k, 2^n)$.

If this path is an internal path, obviously, the noise weight of $(e_1, e_2, \ldots, e_k)$ is not bigger than u. If it is an external path, then $e_k = 1$, and we have the noise weight of $(e_1, e_2, \ldots, e_{k-1}, 0)$ is equal to u. Moreover, the first $k-1$ nodes of other paths with length k are not equal to $f_1 + e_1, f_2 + e_2, \ldots, f_{k-1} + e_{k-1}$. Thus, we can build an injection, by mapping an internal path to $(e_1, e_2, \ldots, e_k)$ and an external path to $(e_1, e_2, \ldots, e_k + 1)$. We have the noise weight of these vectors are not bigger than u. Hence, $\mathcal{N}_k \leq \sum_{i=0}^{u} \binom{k}{u}$. By combining with $\mathcal{N}_k \leq \min(2^k, 2^n)$, we have $\mathcal{N}_k \leq \min(2^k, 2^n, \sum_{i=0}^{u} \binom{k}{u})$

(4) As shown in (2), $\mathcal{N}_m$ is equal to the number of internal paths of length $m-1$, thus we can deduce the conclusion from (3). □

Based on Proposition 7, we have the following theorem about the complexity of $\mathbf{ISBS}_b$ under the assumption about the complexity of incremental solving proposed in Theorem 6.

Theorem 8 *Let $\{f_1, f_2, \ldots, f_m\}$ be the input system of* **ISBS***, and C_k and C are defined as in Theorem 6. Let s be the step size of* $\mathbf{ISBS}_b$*. Suppose $u = rs$ is the noise weight of the optimal solution.*

(a) If the elements in $Im(\mathcal{S}_k)$ are uniformly distributed over $\mathbb{F}_2^k$ for any k, the complexity of $\mathbf{ISBS}_b$ *is*

$$\sum_{t=1}^{r}\sum_{k=1}^{n}\mathcal{N}_k^t(C_k - C_{k-1}) + \sum_{t=1}^{r}\sum_{k=n+1}^{m}\mathcal{N}_k^t C,$$

where $\mathcal{N}_k^t = |Im(\mathcal{S}_k)| \cdot \sum_{i=0}^{ts}\binom{k}{i}/2^k + (|Im(\mathcal{S}_{k-1})| - |Im(\mathcal{S}_k^0)|\binom{k-1}{ts}|/2^{k-1}$, *for any* $1 \le k < m$, *and* $\mathcal{N}_m = |Im(\mathcal{S}_{m-1})| \cdot \sum_{i=0}^{ts}\binom{m-1}{i}/2^{m-1}$.

(b) Otherwise, $\mathbf{ISBS}_b$ *has two complexity bounds*

1. $\sum_{t=1}^{r}\sum_{i=0}^{ts}\binom{m}{i}(C_n + (m-n)C)$,
2. $\sum_{t=1}^{r}\sum_{k=1}^{n}\mathcal{M}_k^t(C_k - C_{k-1}) + \sum_{t=1}^{r}\sum_{k=n+1}^{m-1}\mathcal{M}_k^t C + \sum_{t=1}^{r}\mathcal{M}_{m-1}^t C$,

where $\mathcal{M}_k^t = min(2^n, 2^k, \sum_{i=0}^{ts}\binom{k}{i})$.

Proof: a) When the assumption of uniform distribution holds, the bound can be derived from (1) and (2) of Proposition 7.

b) For an external path $f_1 + e_1, f_2 + e_2, \ldots, f_{k-1} + e_{k-1}$, we can extend the noise vector $(e_1, e_2, \ldots, e_{k-1})$ to a m-dim vector $(e_1, e_2, \ldots, e_{k-1}, 0, \ldots, 0)$ by adding 0 at the end. From the proof of Proposition 7, we can find that for different external paths their corresponding m-dim vectors are distinct , and the noise weights of these m-dim vectors are not bigger than the noise-bound u. Hence, under a noise-bound u, the number of external paths can be bounded by $\sum_{i=0}^{u}\binom{m}{i}$, and the length of these external paths are not greater than m. Therefore, by summing up the complexities with different noise-bounds, we achieve the complexity bound $\sum_{t=0}^{r}\sum_{i=0}^{ts}\binom{m}{i}(C_n + (m-n)C)$

The second bound can be derived from (3) and (4) of Proposition 7 directly. □

Remark 3. When rs is small enough, we have $\sum_{i=0}^{ts}\binom{k}{i} \le 2^n$ and $\sum_{i=0}^{ts}\binom{k}{j} \le 2^k$ for $1 \le t \le r$ and $ts \le k$, which means $\mathcal{M}_k^t = \sum_{i=0}^{ts}\binom{k}{i}$. In this case, by comparing the last bound presented in the Theorem 8 with the complexity bound of **ISBS**, we know that $\mathbf{ISBS}_b$ is much more efficient.

7 The improvements of the backtracking searching process when $m = sn$

From Section 4, we know that the success rate of recovering the true solution increases when m, the number of polynomials, increases. Moreover, for cryptanalysis problems, we have an overdetermined system in most cases, and especially, for some systems, we have $m = sn$, where $s > 1$ is an integer. For example, in the Cold Boot Key recovery problem presented in Section 8, we have $m = rn$ where r is the number of rounds used in the attack. In this section, we will propose a

modification of **ISBS** in the case of $m = sn$, and by this modification we can improve the efficiency of **ISBS** significantly. Note that this modification with $s = 2$ was mentioned in the Appendix of [15]. In this paper we do some theoretical analysis on this modification, and propose some further improvements.

As we mentioned before, when we have a system with n polynomials, in most cases, the number of solutions of this system is very small. For most cryptanalysis problem, we may have the unique solution. Thus, when the first n elements of a noise vector is determined, after we solving the first n polynomials, we will achieve a small zero set $\mathrm{Zero}(\mathbb{P}_n)$. The process of finding a better candidate from $\mathrm{Zero}(\mathbb{P}_n)$ is quite easy. For example, if $\mathrm{Zero}(\mathbb{P}_n)$ has only one point, then the values of the following polynomials will be determined uniquely.

In **ISBS**, when we have a noise-bound u for $(e_1, e_2, \ldots, e_m)$. From the above observation, we know that the algorithm can be divided into two procedures. In the first procedure, we incremental solving the first n polynomials and backtracking search a possible noise vector $(e_1, e_2, \ldots, e_n)$ under noise-bound u, then in the second procedure we achieve the rest $(e_{n+1}, e_{n+2}, \ldots, e_m)$ very fast by solving the remaining polynomials.

Compared with the first procedure, the second procedure is so easy that its complexity can be ignored. Thus we only need to consider a truncate tree which consists of all the paths with length $k \leq n$ in the original search tree. Based on this idea, in this section we will propose a method to significantly reduce the number of external paths if this truncate tree, thus improving the efficiency of **ISBS**.

In order to illustrate this method more clearly, in the following of this section we assume that the input system is a perfectly random system. In this case, no contradiction will occur when dealing with the first n polynomials, and each $\{f_1 + e_1, \ldots, f_n + e_n\}$ has a unique solution, which means for any nodes with depth n, there is one and only one path from it to the external nodes. Under this assumption, similarly as the proof of (b) of Theorem 8, we know that the number of external paths in the truncate tree is equal to $\sum_{i=0}^{u} \binom{n}{i}$. We call an external path in this truncate tree a *branch* for convenience.

7.1 The case m=2n

Lemma 9 *Let u_1, u_2, u be non-negative integers, with $u_1 + u_2 \leq u$. For any non-negative integers a, b, such that $a + b = u - 1$, we have either $u_1 \leq a$ or $u_2 \leq b$.*

Proof: Suppose $u_1 > a$ and $u_2 > b$. Since u_1, u_2 are integers, we have $u_1 \geq a+1$ and $u_2 \geq b+1$. Then $u \geq u_1 + u_2 \geq a + b + 2 = u + 1$, which is a contradiction. □

This lemma shows the following fact. We can divide a noise vector $E = (e_1, \ldots, e_m)$ into two parts $E_1 = (e_1, \ldots, e_n), E_2 = (e_{n+1}, \ldots, e_m)$. If E has a noise-bound u, then either E_1 has a noise-bound a or E_2 has a noise-bound b. We call such noise-bound of E_1 or E_2 a partial noise-bound. For a noise

vector $(e_1, \ldots, e_s)$, we define its *partial noise weight* to be the noise weight of $(e_1, \ldots, e_n)$ when $s > n$, or the noise weight of itself when $s \leq n$. Then, we can build the following *two-direction strategy*. That is first we solve the system from the forward direction, which means we find the optimal solution of the system $\{f_1, f_2, \ldots, f_n, f_{n+1}, \ldots, f_m\}$ under the conditions:

- The partial noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded the partial noise-bound a
- The noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded the total noise-bound u

Then, we solve the system from the backward direction, which means we find the optimal solution of the system $\{f_{n+1}, f_{n+2}, \ldots, f_m, f_1, \ldots, f_n\}$ under the conditions:

- The partial noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded the partial noise-bound b
- The noise weight of $(e_1, e_2, \ldots, e_m)$ is bounded the total noise-bound u

Then the better one of the optimal solutions of the two systems is the optimal solution of the original system. Since the partial noise-bounds a, b can be any integers, thus an important problem about improving the efficiency of this strategy is how to choose a and b such that the total number of branches for the two systems is minimum.

Lemma 10 *Let a, b, u, n be non-negative integers, with $n/2 \geq u$ and $a + b = u - 1$. Then we have:*

$$\sum_{i=0}^{u} \binom{n}{i} > \sum_{i=0}^{a} \binom{n}{i} + \sum_{i=0}^{b} \binom{n}{i} \geq \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i}$$

Proof: Without loss of generality, we can assume $a \leq b$. First we prove $\sum_{i=0}^{u} \binom{n}{i} > \sum_{i=0}^{a} \binom{n}{i} + \sum_{i=0}^{b} \binom{n}{i}$. It is equal to prove $\sum_{i=a+1}^{u} \binom{n}{i} > \sum_{i=0}^{b} \binom{n}{i}$. Note that the numbers of terms in two sides of the inequality are both $b + 1$. Moreover, since $n/2 \geq u \geq a + 1 > 0$, we have $\binom{n}{a+1} > \binom{n}{0}$, $\binom{n}{a+2} > \binom{n}{1}$, $\binom{n}{u} > \binom{n}{b}$. By summing up all these inequalities, we have $\sum_{i=a+1}^{u} \binom{n}{i} > \sum_{i=0}^{b} \binom{n}{i}$.

Now we prove the second inequality. Obviously, when $a = \lfloor \frac{a+b}{2} \rfloor$, which means that either u is odd and $a = b$ or u is even and $a = b - 1$, the equality holds. Without loss of generality, we can assume $a < \lfloor \frac{a+b}{2} \rfloor$, which implies $a < \lfloor \frac{a+b}{2} \rfloor \leq \lceil \frac{a+b}{2} \rceil < b$. Then it is sufficient to show that

$$\binom{n}{a+1} + \binom{n}{a+2} + \cdots + \binom{n}{\lfloor \frac{a+b}{2} \rfloor} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 1} + \binom{n}{\lceil \frac{a+b}{2} \rceil + 2} + \cdots + \binom{n}{b}.$$

Since $n/2 \geq u \geq b$, we have the numbers of terms in both sides of the above inequality are same, and $\binom{n}{a+1} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 1}, \binom{n}{a+2} < \binom{n}{\lceil \frac{a+b}{2} \rceil + 2}, \ldots, \binom{n}{\lfloor \frac{a+b}{2} \rfloor} < \binom{n}{b}$. Therefore, we can deduce our conclusion by summing up all these inequalities. □

Corollary 11 *Let a, b, u, n be non-negative integers, with $n/2 \geq u$ and $a + b = u - 1$. Let T be a positive constant. Then we have:*

$$\sum_{i=0}^{u} \binom{n}{i} T > \sum_{i=0}^{a} \binom{n}{i} T + \sum_{i=0}^{b} \binom{n}{i} T \geq \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} T + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i} T$$

Remark 4. When the condition $n/2 \geq u$ doesn't hold, the conclusion of Lemma 10 is still valid. The interested reader is referred to the appendix of [15] for detailed proof. In this paper, we only consider the case of $n/2 \geq u$ which holds for most Max-PoSSo problems.

Lemma 10 shows that given a noise-bound u, the branches we need to solve in the two-direction strategy is less than those in the one-direction strategy. If the running times for solving polynomial systems from different directions are the same, from Corollary 11, we can conclude that the best scheme is using the partial noise-bound $\lfloor \frac{a+b}{2} \rfloor$ for one system and $\lceil \frac{a+b}{2} \rceil$ for another system. For example, let $n = 128$, $u = 10$. If we consider the random system, for the one-direction strategy we need to solve $\sum_{i=0}^{10} \binom{128}{i} \approx 2^{47.8}$ branches. For the optimal two-direction strategy we need to solve $\sum_{i=0}^{4} \binom{128}{i} + \sum_{i=0}^{5} \binom{128}{i} \approx 2^{28.1}$ branches. Obviously, this is a significant improvement.

From Section 3.3, we know that for different polynomial order, the efficiency of incrementally solving is different. Thus, when we use the two-direction strategy, the times for solving the two systems are different. Hence, we should modification the above conclusions under more specific conditions, and we have the following lemma.

Theorem 12. *Let a, b, u, n be non-negative integers, with $n/2 \geq u$ and $a + b = u - 1$. Let $T_1 \leq T_2$ be two positive constants. If $T_2/T_1 \leq \binom{n}{\lceil \frac{a+b}{2} \rceil + 1} / \binom{n}{\lfloor \frac{a+b}{2} \rfloor} = \binom{n}{\lceil \frac{u-1}{2} \rceil + 1} / \binom{n}{\lfloor \frac{u-1}{2} \rfloor}$, then we have:*

1. $\sum_{i=0}^{a} \binom{n}{i} T_2 + \sum_{i=0}^{b} \binom{n}{i} T_1 \geq \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} T_2 + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i} T_1$
2. $\sum_{i=0}^{u} \binom{n}{i} T_1 > \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} T_2 + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i} T_1$

Proof: First, we compare $A = \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor} \binom{n}{i} T_2 + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil} \binom{n}{i} T_1$ with $B_1 = \sum_{i=0}^{\lfloor \frac{a+b}{2} \rfloor + 1} \binom{n}{i} T_2 + \sum_{i=0}^{\lceil \frac{a+b}{2} \rceil - 1} \binom{n}{i} T_1$. Then $B_1 - A = \binom{n}{\lfloor \frac{a+b}{2} \rfloor + 1} T_2 - \binom{n}{\lceil \frac{a+b}{2} \rceil} T_1$. Since $\lfloor \frac{a+b}{2} \rfloor + 1 \geq \lceil \frac{a+b}{2} \rceil$ and $T_2 \geq T_1$, we have $B_1 - A > 0$.

Let B_t be $\sum_{i=0}^{\lfloor\frac{a+b}{2}\rfloor+t}\binom{n}{i}T_2+\sum_{i=0}^{\lceil\frac{a+b}{2}\rceil-t}\binom{n}{i}T_1$. For any $2\le t\le\lceil\frac{a+b}{2}\rceil$, we have $B_t-B_{t-1}=\binom{n}{\lfloor\frac{a+b}{2}\rfloor+t}T_2-\binom{n}{\lceil\frac{a+b}{2}\rceil-t+1}T_1$. Since $\lfloor\frac{a+b}{2}\rfloor+t\le a+b<u\le n/2$, we have $\binom{n}{\lfloor\frac{a+b}{2}\rfloor+t}>\binom{n}{\lceil\frac{a+b}{2}\rceil-t+1}$ and $T_2\ge T_1$. Therefore, $B_t-B_{t-1}>0$ holds, which implies $A\le B_t$ for any $2\le t\le\lfloor\frac{a+b}{2}\rfloor$.

Now let C_1 be $\sum_{i=0}^{\lfloor\frac{a+b}{2}\rfloor-1}\binom{n}{i}T_2+\sum_{i=0}^{\lceil\frac{a+b}{2}\rceil+1}\binom{n}{i}T_1$. $C_1-A=\binom{n}{\lceil\frac{a+b}{2}\rceil+1}T_1-\binom{n}{\lfloor\frac{a+b}{2}\rfloor}T_2$. From the hypothesis $T_2/T_1\le\binom{n}{\lceil\frac{a+b}{2}\rceil+1}/\binom{n}{\lfloor\frac{a+b}{2}\rfloor}$, we can deduce $C_1\ge A$.

Similarly, we can define C_t to be $\sum_{i=0}^{\lfloor\frac{a+b}{2}\rfloor-t}\binom{n}{i}T_2+\sum_{i=0}^{\lceil\frac{a+b}{2}\rceil+t}\binom{n}{i}T_1$. Then $C_t-C_{t-1}=\binom{n}{\lceil\frac{a+b}{2}\rceil+t}T_1-\binom{n}{\lfloor\frac{a+b}{2}\rfloor-t+1}T_2$. Obviously, when $1<t<u-\lceil\frac{a+b}{2}\rceil$, we have $\binom{n}{\lceil\frac{a+b}{2}\rceil+t}>\binom{n}{\lceil\frac{a+b}{2}\rceil+1}$ and $\binom{n}{\lfloor\frac{a+b}{2}\rfloor-t+1}<\binom{n}{\lfloor\frac{a+b}{2}\rfloor}$. It means that

$$\binom{n}{\lceil\frac{a+b}{2}\rceil+t}\Big/\binom{n}{\lfloor\frac{a+b}{2}\rfloor-t+1}>\binom{n}{\lceil\frac{a+b}{2}\rceil+1}\Big/\binom{n}{\lfloor\frac{a+b}{2}\rfloor}\ge T_2/T_1.$$

Hence, we have $C_t>C_{t-1}$ for any $2\le t\le\lfloor\frac{a+b}{2}\rfloor$, which implies $C_t\ge A$. From $A\le B_t$ and $A\le C_t$, we can derive the first inequality of the theorem.

From the above proof, we know that $A<C_{\lfloor\frac{a+b}{2}\rfloor}=T_2+\sum_{i=0}^{u-1}\binom{n}{i}T_1$. Then, we have $\sum_{i=0}^{u}\binom{n}{i}T_1-C_{\lfloor\frac{a+b}{2}\rfloor}=\binom{n}{u}T_1-T_2>0$, since $T_2/T_1\le\binom{n}{\lceil\frac{a+b}{2}\rceil+1}/\binom{n}{\lfloor\frac{a+b}{2}\rfloor}<\binom{n}{u}$. This proves the second inequality of the theorem. □

Theorem 12 shows the following fact. Suppose that the average time for solving one branches from the forward direction is T_1, and the average time for solving one branches from the backward direction is T_2. Assume the polynomial system corresponding to the forward direction has the best polynomial ordering, then $T_1\le T_2$. Moreover, if $T_2/T_1\le\binom{n}{\lceil\frac{u-1}{2}\rceil+1}/\binom{n}{\lfloor\frac{u-1}{2}\rfloor}$, where u is the total noise-bound. Then the best scheme is setting the partial noise-bound to be $\lceil\frac{u-1}{2}\rceil$ for the forward direction, and $\lfloor\frac{u-1}{2}\rfloor$ for the backward direction.

Furthermore, from the proof of Theorem 12, we know that if

$$\binom{n}{\lceil\frac{u-1}{2}\rceil+t}\Big/\binom{n}{\lfloor\frac{u-1}{2}\rfloor-t+1}<T_2/T_1\le\binom{n}{\lceil\frac{u-1}{2}\rceil+t+1}\Big/\binom{n}{\lfloor\frac{u-1}{2}\rfloor-t},$$

the best scheme is setting the partial noise-bound to be $\lceil\frac{u-1}{2}\rceil+t$ for the forward direction, and $\lfloor\frac{u-1}{2}\rfloor-t$ for the backward direction.

Based on the above two-direction strategy and the artificial noise-bound strategy proposed in Section 6, we present the following algorithm $\mathbf{ISBS}_2$.

Algorithm 3: ISBS$_2$ algorithm

input : A sorted Boolean polynomial sets $\mathbb{P}_0 = \{f_1, f_2, \ldots, f_m\}$ with n variables, where $m = 2n$;
An integer G which is the step size;

output: $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ s.t. $\{i \in N | f_i(x_1, \ldots, x_n) = 0, f_i \in \mathbb{P}_0\}$ has maximal cardinality;

$\mathbb{P}_1 \leftarrow \{f_{n+1}, f_{n+2}, \ldots, f_m, f_1, f_2, \ldots, f_n\}$;
Let t_0, u_0, u_{p_0}, $E = [e_1, e_2, \ldots, e_m], \mathbb{QS} = \{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}$ be the output of $\texttt{Candidate}_2(\mathbb{P}_0, \emptyset, m, 0, n, 0)$;
Let t_1, u_1, u_{p_1}, $B = [b_1, b_2, \ldots, b_m], \mathbb{RS} = \{\mathbb{R}_0, \mathbb{R}_1, \ldots, \mathbb{R}_m\}$ be the output of $\texttt{Candidate}_2(\mathbb{P}_1, \emptyset, m, 0, n, 0)$; `/* Here` $t_0 = t_1 = m$ `*/`
Let T_0 be the running time of $\texttt{Candidate}_2(\mathbb{P}_0, \emptyset, m, 0, n, 0)$, and T_1 be the running time of $\texttt{Candidate}_2(\mathbb{P}_1, \emptyset, m, 0, n, 0)$;
$r \leftarrow T_1/T_0$; `/* The time ratio */`
$u_m \leftarrow \min(u_0 - 1, u_1 - 1)$, $u_s \leftarrow 0$;
if $u_m = u_0 - 1$ **then** $\mathbb{S} = \mathbb{Q}_m$**else** $\mathbb{S} = \mathbb{R}_m$
while $u_s < u_m$ **do**
    $u_s \leftarrow \min(u_s + G, u_m)$;
    $u_{s_0} \leftarrow \texttt{PartialBound}(u_s, n, 0, r)$;
    Let $\mathbb{S}_0$ and u_s be the output of $\texttt{Backtracking}_2(\mathbb{P}_0, E, \mathbb{QS}, u_s, u_0, u_{s_0}, u_{p_0}, 0, r)$;
    $u_{s_1} \leftarrow \texttt{PartialBound}(u_s, n, 1, r)$;
    Let $\mathbb{S}_1$ and u_s be the output of $\texttt{Backtracking}_2(\mathbb{P}_1, B, \mathbb{RS}, u_s, u_1, u_{s_1}, u_{p_1}, 1, r)$;
    if $\mathbb{S}_1 \neq \emptyset$ **then**
        $\mathbb{S} \leftarrow \mathbb{S}_1$ and **break**;
    else if $\mathbb{S}_0 \neq \emptyset$ **then**
        $\mathbb{S} \leftarrow \mathbb{S}_0$ and **break**;
Get $(x_1, \ldots, x_n)$ from $\mathbb{S}$, and **return** $(x_1, \ldots, x_n)$.

Function: `Candidate`$_2$

input : $\mathbb{P} = \{f_{s+1}, f_{s+2}, \ldots, f_m\}$: a polynomial set with n variables,
`Result`($\mathbb{R}$) : the common zero of a polynomial set $\mathbb{R}$,
u_{bound}: a noise-bound;
u: the noise weight of $(e_1, \ldots, e_s)$;
$u_{partial}$: a partial noise-bound;
u_p: the partial noise weight of $(e_1, \ldots, e_s)$;

output: t: an index number;
u: the noise weight of $(e_1, \ldots, e_{s+t})$;
u_p: the partial noise weight of $(e_1, \ldots, e_{s+t})$;
$\{\mathbb{Q}_s, \mathbb{Q}_{s+1}, \ldots, \mathbb{Q}_{s+t}\}$:a sequence of zero sets;
$E = (e_{s+1}, \ldots, e_{s+t})$: a vector;

$\mathbb{Q}_s \leftarrow$ `Result`($\mathbb{R}$);
for i *from* $s+1$ *to* m **do**
  $\mathbb{P}_i \leftarrow \{\mathbb{Q}_{i-1}, f_i\}$, solve $\mathbb{P}_i$ and achieve `Result`($\mathbb{P}_i$);
  if *Result*($\mathbb{P}_i$) $= \{1\}$ **then**
    $\mathbb{Q}_i \leftarrow \mathbb{Q}_{i-1}$,$e_i \leftarrow 1$, $u \leftarrow u+1$;
    if $i \le n$ **then** $u_p = u_p + 1$
    if $u > u_{bound}$ *or* $u_p > u_{partial}$ **then**
      $t \leftarrow i - s$ and **break;**
  else if *Result*($\mathbb{P}_i$) *and* $\mathbb{Q}_{i-1}$ *have the same zero set* **then**
    $\mathbb{Q}_i \leftarrow \mathbb{Q}_{i-1}$, $e_i = 0$;
  else
    $\mathbb{Q}_i \leftarrow$ `Result`($\mathbb{P}_i$), $e_i = 0$;
return t, u, u_p,$\{\mathbb{Q}_s, \mathbb{Q}_{s+1}, \ldots, \mathbb{Q}_{s+t}\}$ and $(e_{s+1}, e_{s+2}, \ldots, e_{s+t})$.

Function: `PartialBound`

input : u: a total noise-bound
n: the number of variables
d: a Boolean value indicating the direction of searching
r: time ratio.

output: u_p: a partial noise-bound

Find the t s.t. $\binom{n}{\lceil \frac{u-1}{2} \rceil + t} / \binom{n}{\lfloor \frac{u-1}{2} \rfloor - t + 1} < r \le \binom{n}{\lceil \frac{u-1}{2} \rceil + t + 1} / \binom{n}{\lfloor \frac{u-1}{2} \rfloor - t}$;
if $d = 0$ **then return** $\lceil \frac{u-1}{2} \rceil + t$
else return $\lfloor \frac{u-1}{2} \rfloor - t$

Function: $\texttt{Backtracking}_2$

input : $\mathbb{P} = \{f_1, f_2, \ldots, f_m\}$: a polynomial set with n variables,
$E = (e_1, e_2, \ldots, e_m)$: a noise vector,
$\{\mathbb{Q}_0, \mathbb{Q}_1, \ldots, \mathbb{Q}_m\}$: a sequence of zero sets;
u_{bound}: a noise-bound;
u: the noise weight of E
$u_{partial}$: a partial noise-bound;
u_p: the partial noise weight of E
d: a Boolean value indicating the forward or backward direction
r: time ratio.

output: A zero set $\mathbb{S}$ and a noise-bound u_{bound}

$k \leftarrow m$, $\mathbb{S} \leftarrow \emptyset$; `/* k is the backtracking index */`
while $k \geq 1$ **do**
  if $k \leq n$ **then** $c \leftarrow 1$**else** $c \leftarrow 0$
  if $e_k = 0$, $\mathbb{Q}_k \neq \mathbb{Q}_{k-1}$, $u + 1 \leq u_{bound}$ *and* $u_p + c \leq u_{partial}$ **then**
    $e_k \leftarrow 1$, $u \leftarrow u + 1$, $u_p \leftarrow u_p + c$;
    Solve $\mathbb{P}_k = \{\mathbb{Q}_{k-1}, f_k + 1\}$ and achieve $\texttt{Result}(\mathbb{P}_k)$;
    Set $\mathbb{Q}_k = \texttt{Result}(\mathbb{P}_k)$;
    Let t, u, u_p, $\{\mathbb{Q}_k, \mathbb{Q}_{k+1}, \ldots, \mathbb{Q}_{k+t}\}$, $(e_{k+1}, \ldots, e_{k+t})$ be the output of $\texttt{Candidate}_2(\{f_{k+1}, \ldots, f_m\}, \mathbb{Q}_k, u_{bound}, u, u_{partial}, u_p)$;
    $k \leftarrow k + t$;
    if $u \leq u_{bound}$ *and* $u_p \leq u_{partial}$ **then** `/* In this case, k = m */`
      $\mathbb{S} \leftarrow \mathbb{Q}_m$, $u_{bound} \leftarrow u - 1$;
      $u_{partial} \leftarrow \texttt{PartialBound}(u_{bound}, r, d)$;
  else
    $u \leftarrow u - e_k$, $u_p \leftarrow u_p - c \cdot e_k$, $k \leftarrow k - 1$;
return $\mathbb{S}$ and u_{bound}.

For the complexity of $\mathbf{ISBS}_2$, we have the following theorem.

Theorem 13 *Let $\{f_1, f_2, \ldots, f_n, f_{n+1}, f_{n+2}, \ldots, f_{2n}\}$ be the input polynomial system of $\mathbf{ISBS}_2$, and C_k and C are defined are defined as in Theorem 6. Let s be the step size of $\mathbf{ISBS}_2$, and $u = rs$ be the noise weight of the optimal solution. If $\{f_1, f_2, \ldots, f_n, f_{n+1}, f_{n+2}, \ldots, f_{2n}\}$ and $\{f_{n+1}, f_{n+2}, \ldots, f_{2n}, f_1, f_2, \ldots, f_n\}$ are both perfectly random systems, and the complexity of solving the polynomials systems from two directions are the same, then the complexity of $\mathbf{ISBS}_2$ is*

$$(\sum_{t=1}^{r} \sum_{i=0}^{\lfloor (ts-1)/2 \rfloor} \binom{n}{i} + \sum_{t=1}^{r} \sum_{i=0}^{\lceil (ts-1)/2 \rceil} \binom{n}{i})(C_n + (m-n)C)$$

Proof: Since we assume the complexity of solving from two directions are the same, we will set the partial noise-bounds to be $\lfloor (u-1)/2 \rfloor$ and $\lceil (u-1)/2 \rceil$, when the total noise-bound is u. Without loss of generality, suppose $\lfloor (u-1)/2 \rfloor$ is set to be the partial noise-bound of $\{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$. Since this polynomial system is perfectly random, for any node with depth n in the search tree, there is a unique path from it to the terminal nodes, which means the

number of the external paths of this tree is equal to the number of the external paths of its truncate searching tree. Thus, when the partial noise-bound is $\lfloor (u-1)/2 \rfloor$, the number of external paths is $\sum_{i=0}^{\lfloor (u-1)/2 \rfloor} \binom{n}{i}$. Hence the complexity of finding the optimal solution of $\{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$ by $\mathbf{ISBS}_2$ under total noise-bound u and partial noise-bound $\lfloor (u-1)/2 \rfloor$, is$\sum_{i=0}^{\lfloor (u-1)/2 \rfloor} \binom{n}{i}(C_n + (m-n)C)$. Therefore, by summing up the complexities of finding the optimal solution of $\{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}\}$ and $\{f_{n+1}, \ldots, f_{2n}, f_1, \ldots, f_n\}$ under increasing noise-bounds, we have the conclusion. □

Remark 5. When $m \neq 2n$, $\mathbf{ISBS}_2$ can work with slight modification. The difference is in this case the partial noise-bound is w.r.t. the first $m/2$ polynomials. In this case, the complexity of solving the whole system is not dominated by the complexity of solving the first $m/2$ polynomials, thus the efficiency improvement will not be as significant as that in the case of $m = 2n$.

7.2 The Optimal Division Strategy

In the above section, we divide the input system with $2n$ elements into two sub-systems with n elements. A natural problem for $\mathbf{ISBS}_2$ is that if we divide the input system into one system with $n-k$ elements and another system with $n+k$ elements, whether the algorithm will be better. In this section, we will talk about this problem.

For convenience, in this section, when we say a k-partial noise-bound, we mean the noise-bound about the noise vector $(e_1, e_2, \ldots, e_k)$. Then, the above division strategy is:

(A) Given a total noise-bound u, we will find the optimal solution from $\mathbb{P}_1 = \{f_1, f_2, \ldots, f_{n-k}, f_{n-k+1}, \ldots, f_{2n}\}$ under a $(n-k)$-partial noise-bound a and $\mathbb{P}_2 = \{f_{n-k+1}, f_{n-k+2}, \ldots, f_{2n}, f_1, f_2, \ldots, f_{n-k}\}$ under a $(n+k)$-partial noise-bound b , where $a + b = u - 1$.

The division strategy proposed in last section is:

(B) Given a total noise-bound u, we will find the optimal solution from $\mathbb{P}_1 = \{f_1, f_2, \ldots, f_{n-k}, f_{n-k+1}, \ldots, f_{2n}\}$ under a n-partial noise-bound a and $\mathbb{P}_3 = \{f_{n+1}, f_{n+2}, \ldots, f_{2n}, f_1, f_2, \ldots, f_n\}$ under a n-partial noise-bound b , where $a + b = u - 1$.

In the following we will prove that when $k > 0$ Strategy B is always better than Strategy A, if $\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3$ are perfectly random systems and the complexity of incremental solving are the same when the order of the polynomials is changed.

Firstly, let's consider the forward direction. That is we deal with $\mathbb{P}_1$ in Strategy A and B. We show that each external path exists in the search tree of Strategy B will be in the search tree of Strategy A. Since $\mathbb{P}_1$ is a perfect random system, the number of the external paths of its search tree is equal to that of its truncate search tree with depth n. When we incremental solving all possible $\{f_1 + e_1, f_2 + e_2, \ldots, f_{n-k} + e_{n-k}\}$, since the $(n-k)$-partial

noise-bound of this system is a, the number of paths generated from the root with depth $n-k$ is $\binom{n-k}{a}$. Then when we go on to incrementally solve all possible $\{f_{n-k+1}+e_{n-k+1},\ldots,f_n+e_n\}$, the only constrain is that the noise weight of $(e_1,\ldots,e_n)$ should be bounded by u. Therefore, if the noise weight of $(e_1,\ldots,e_{n-k})$ is u_1, we have to keep the noise weight of $(e_{n-k+1},\ldots,e_n)$ no bigger than $u-u_1$. Hence, the number of external paths of the truncate search tree with depth n is equal to $\sum_{i=0}^{a}\binom{n-k}{i}\sum_{j=0}^{u-i}\binom{k}{j}$. We have the following inequality about this value.

Lemma 14 $\sum_{i=0}^{a}\binom{n-k}{i}\sum_{j=0}^{u-i}\binom{k}{j}\geq\sum_{i=0}^{a}\binom{n}{i}$

Proof: By Vandermonde's identity, we have

$$\begin{aligned}
\tbinom{n}{a} &= \tbinom{n-k}{a}\tbinom{k}{0}+\tbinom{n-k}{a-1}\tbinom{k}{1}+\cdots+\tbinom{n-k}{0}\tbinom{k}{a}\\
\tbinom{n}{a-1} &= \tbinom{n-k}{a-1}\tbinom{k}{0}+\tbinom{n-k}{a-2}\tbinom{k}{1}+\cdots+\tbinom{n-k}{0}\tbinom{k}{a-1}\\
&\vdots\\
\tbinom{n}{0} &= \tbinom{n-k}{0}\tbinom{k}{0}.
\end{aligned}$$

Since $u-i+i=u\geq a$, the terms in the right of the above equalities are all in $\sum_{i=0}^{a}\binom{n-k}{i}\sum_{j=0}^{u-i}\binom{k}{j}$, and these terms are distinct. Thus, the inequality is valid. □

Note that $\sum_{i=0}^{a}\binom{n}{i}$ is equal to the number of external paths in the search tree of Strategy B. Since the change of the partial noise-bound before depth n doesn't effect the incremental solving process after depth n, which means the complexity of incremental solving the external paths of different strategies are the same. Hence, by Lemma 14, we can deduce that Strategy B is better than Strategy A for the forward direction.

Secondly, let's consider the backward direction. That is we deal with $\mathbb{P}_2$ in Strategy A, and $\mathbb{P}_3$ in Strategy B. Similarly as above, we consider the truncate search tree of $\mathbb{P}_2$ with depth n. For $\mathbb{P}_2$, we have a $(n+k)$-partial noise-bound b, when we are incremental solving all the possible $\{f_{n+1}+e_1,\ldots,f_{2n}+e_n\}$. Therefore we should keep the noise weight of $(e_1,\ldots,e_n)$ no bigger than b. Hence the number of external paths of this truncate search tree is $\sum_{i=0}^{b}\binom{n}{i}$. Note that, for Strategy B the number of external paths of the truncate search tree is also $\sum_{i=0}^{b}\binom{n}{i}$. Since $\mathbb{P}_2$ and $\mathbb{P}_3$ are perfectly random system, we can deduce that the numbers of external paths of the two search trees are the same.

For the two search trees, the only difference is that in the tree of Strategy A, some external paths may have depth $n+d_1$, where $0<d_1\leq k$, while in the tree of Strategy B, the corresponding external paths may have depth $n+d_2$, where $d_1<d_2$. The reason is that, in Strategy A, there may be some noise vectors $(e_1,e_2,\ldots,e_{n+d_1})$ such that the noise weight of $(e_1,e_2,\ldots,e_n)$ is bound by b, but that of $(e_1,e_2,\ldots,e_{n+d_1})$ exceeds b. In Strategy B, since we don't constrain the noise weight of $(e_1,e_2,\ldots,e_{n+k})$, this path will be pruned only when the

the noise weight exceeds the total noise-bound u. From our assumption of the complexity of incremental solving, we know that C is much smaller than C_n, thus we have $C_n + d_1C \approx C_n + d_2C$, which means the complexity of solving these external paths are almost the same. Hence, Strategy A and Strategy B are almost equivalence for the backward direction.

By combining the conclusions of the two directions, we can conclude that Strategy B is always the optimal strategy.

For the practical problems, the input systems are not perfectly random, and theoretically finding the optimal is nearly impossible. In this case, from the above proof, we can know that Strategy B is still a very good strategy, when the assumption of incremental solving is valid.

7.3 The case $m = sn$

In this section, we present some theoretical results by which we can show that the two-direction strategy can be extended to a s-direction strategy when $m = sn$.

Similarly as Lemma 9, we have the following lemma.

Lemma 15 *Let $u_1, u_2, \ldots, u_s, u$ be $s+1$ non-negative integers, with $u_1 + u_2 + \cdots + u_s \leq u$. For any non-negative integers $a_1, a_2, \ldots, a_s$, such that $a_1 + a_2 + \cdots + a_s = u - 1$, we have at least one of the following inequalities hold $u_1 \leq a_1$, $u_2 \leq a_2$,..., $u_s \leq a_s$.*

This lemma shows the following fact. We can divide a noise vector $E = (e_1, \ldots, e_m)$ into s parts

$$E_1 = (e_1, \ldots, e_n), E_2 = (e_{n+1}, \ldots, e_{2n}), \ldots, E_s = (e_{(s-1)n+1}, \ldots, e_{sn})$$

If E has a noise-bound u, then there is a E_i whose noise-bound is a_i. Thus, we can build an s-direction searching strategy like the case $m = 2n$. That is solving the system $\{f_1, f_2, \ldots, f_n, \ldots\}$ under the partial noise-bound a_1 and total noise-bound u, solving the system $\{f_{n+1}, f_{n+2}, \ldots, f_{2n}, \ldots\}$ under the partial noise-bound a_2 and total noise-bound u, ... , solving the system $\{f_{m-s+1}, f_{m-s+2}, \ldots, f_m, \ldots\}$ under the partial noise-bound a_s and total noise-bound u. Now, let's show the best scheme to set the values of these a_i's.

Theorem 16. *Let u, n be two non-negative integers with $n/2 \geq u$. $s \geq 2$ is an integer. Suppose $u - s + 1 \equiv r \bmod s$, and $p = (u - s + 1 - r)/s$. $a_1, a_2, \ldots, a_s$ are s non-negative integers, s.t. $a_1 + a_2 + \cdots + a_s = u - s + 1$. We have*

$$\sum_{i=0}^{u} \binom{n}{i} > \sum_{j=1}^{s} \sum_{i=0}^{a_j} \binom{n}{i} \tag{1}$$

$$\sum_{j=1}^{s} \sum_{i=0}^{a_j} \binom{n}{i} \geq \sum_{j=1}^{s-r} \sum_{i=0}^{p} \binom{n}{i} + \sum_{j=1}^{r} \sum_{i=0}^{p+1} \binom{n}{i} \tag{2}$$

Proof: First, let's prove inequality (1). Since $u+1=(a_1+1)+(a_2+1)+\cdots+(a_s+1)$, we can divide $\sum_{i=0}^{u}\binom{n}{i}$ into s parts :

$$\tbinom{n}{0}+\ldots+\tbinom{n}{a_1},\ \tbinom{n}{a_1+1}+\ldots+\tbinom{n}{a_1+a_2+1},\ldots,\ \tbinom{n}{u-a_s}+\ldots+\tbinom{n}{u}.$$

Note that the j-th part is the sum of a_j elements. In the following, when we say a left-part, we mean one of these parts. We can write $\sum_{j=0}^{s}\sum_{i=0}^{a_j}\binom{n}{i}$ as the sum of s parts $\sum_{i=0}^{a_1}\binom{n}{i},\ldots,\sum_{i=0}^{a_s}\binom{n}{i}$. In the following, when we say a right-part, we mean one of such parts. Obviously the first left-part $\sum_{i=0}^{a_1}\binom{n}{i}$ is equal to the first right-part. For the j-th left-part with $j>1$, since $n/2\geq u$, we can check that each element in this left-part is bigger than the corresponding element in the j-th right-part. Hence the j-th left-part is bigger than the j-th right-part. Since $s\geq 2$, we at least have two parts. Thus, we can derive the first inequality.

Now, let's prove inequality (2). Without loss of generality, we can assume $a_1\leq a_2\leq\cdots\leq a_s$ Note that $a_1+a_2+\cdots+a_s=u-s+1=(s-r)p+r(p+1)$. Suppose among these a_i, there are s_1 elements being smaller than p, b elements being equal to p, c elements being equal to $p+1$ and s_2 elements being bigger than $p+1$. Then $a_1,a_2,\ldots,a_s$ can be written as

$$a_1,a_2,\ldots,a_{s_1},\underbrace{p,p,\ldots,p}_{b},\underbrace{p+1,p+1,\ldots,p+1}_{c},a_{s-s_2+1},a_{s-s_2+2},\ldots,a_s,$$

where $s_1+b+c+s_2=s$

Now we consider the following three cases:

1. $s_1+b=s-r$ and $c+s_2=r$. In this case, the left side of (2) minus the right side of (2) is equal to

$$\sum_{j=1}^{s_2}\sum_{i=p+2}^{a_{s-s_2+j}}\binom{n}{i}-\sum_{j=1}^{s_1}\sum_{i=a_j+1}^{p}\binom{n}{i} \tag{3}$$

   Since $a_1+a_2+\cdots+a_s+s=(s-r)(p+1)+r(p+2)$, the first and second parts of (3) have the same number of terms. Moreover, $a_{s-s_2+j}\geq p+2>p\geq a_i+1$, which means every term in the left part of (3) is bigger than that in the right part of (3). Then we have (3) is not smaller than 0, which implies that the second inequality of (2) is valid, and the equality holds when $b=s-r$ and $c=r$.
2. $s_1+b>s-r$ and $c+s_2<r$. In this case, the left side of (2) minus the right side of (2) is equal to

$$\sum_{j=1}^{s_2}\sum_{i=p+2}^{a_{s-s_2+j}}\binom{n}{i}-\Big(\sum_{j=1}^{b+s_1-s+r}\binom{n}{p+1}+\sum_{j=1}^{s_1}\sum_{i=a_j+1}^{p}\binom{n}{i}\Big) \tag{4}$$

   Similarly as case 1, the first and second parts of (3) have the same number of terms, and every term in the left is bigger than that in the right, which implies the correctness of the second inequality of (2).

3. $s_1 + b < s - r$ and $c + s_2 > r$. In this case, then the left side of (2) minus the right side of (2) is equal to

$$(\sum_{j=1}^{s_2} \sum_{i=p+2}^{a_{s-s_2+j}} \binom{n}{i} + \sum_{j=1}^{c+s_2-r} \binom{n}{p+1}) - \sum_{j=1}^{s_1} \sum_{i=a_j+1}^{p} \binom{n}{i} \tag{5}$$

Similarly as the above cases, the first and second parts of (5) have the same number of terms, and every term in the left is bigger than that in the right, which implies the correctness of the second inequality of (2).

In summary, inequality (2) is valid in any cases. □

This theorem shows that when the times for solving different systems are equal, s-direction method is always better than one-direction method, and the best strategy is setting the noise-bounds of $s - r$ systems to be p and those of the rest r systems to be $p + 1$.

The next theorem shows that when the time ratios for solving different systems are bounded by some values, the best strategy is setting the partial noise-bounds of the $s - r$ harder systems to be p and those of the rest r easier ones to be $p + 1$.

Theorem 17. *Let u, n be two non-negative integers with $n/2 \geq u$. $s \geq 2$ is an integer. Suppose $u - s + 1 \equiv r \bmod s$, and $p = (u - s + 1 - r)/s$. $a_1, a_2, \ldots, a_s$ are s non-negative integers, s.t. $a_1 + a_2 + \cdots + a_s = u - s + 1$. Let $T_1 \geq T_2 \geq \cdots \geq T_s > 0$ be s constants. If $T_1/T_s \leq \binom{n}{\lceil (u-s+1)/s \rceil + 1} / \binom{n}{\lfloor (u-s+1)/s \rfloor}$, $T_1/T_{s-r} \leq \binom{n}{\lfloor (u-s+1)/s \rfloor + 1} / \binom{n}{\lfloor (u-s+1)/s \rfloor}$, $T_{s-r+1}/T_s \leq \binom{n}{\lceil (u-s+1)/s \rceil + 1} / \binom{n}{\lceil (u-s+1)/s \rceil}$, then*

$$\sum_{j=1}^{s} \sum_{i=0}^{a_j} \binom{n}{i} T_j \geq \sum_{j=1}^{s-r} \sum_{i=0}^{p} \binom{n}{i} T_j + \sum_{j=1}^{r} \sum_{i=0}^{p+1} \binom{n}{i} T_j \tag{6}$$

$$\sum_{i=0}^{u} \binom{n}{i} T_s > \sum_{j=1}^{s-r} \sum_{i=0}^{p} \binom{n}{i} T_j + \sum_{j=1}^{r} \sum_{i=0}^{p+1} \binom{n}{i} T_j \tag{7}$$

Proof: Since the full proof of inequality (6) is complicated, here we only prove the most typical step. We can assume $r > 0$, since for the case $r = 0$, the result can be proved similarly. Then we have $\lceil (u - s + 1)/s \rceil + 1 = p + 2$ and $\lfloor (u - s + 1)/s \rfloor = p$. Let $A = \sum_{j=1}^{s-r} \sum_{i=0}^{p} \binom{n}{i} T_j + \sum_{j=1}^{r} \sum_{i=0}^{p+1} \binom{n}{i} T_j$. Now we consider the case of $a_1, a_2, \ldots, a_s$ being equal to

$$p - 1, \underbrace{p, p, \ldots, p}_{s-r-1}, \underbrace{p + 1, p + 1, \ldots, p + 1}_{r-1}, p + 2$$

We can set $B_1 = \sum_{i=0}^{p-1} \binom{n}{i} T_1 + \sum_{j=1}^{s-r-1} \sum_{i=0}^{p} \binom{n}{i} T_j + \sum_{j=1}^{r-1} \sum_{i=0}^{p+1} \binom{n}{i} T_j + \sum_{i=0}^{p+2} \binom{n}{i} T_s$. Then by the hypothesis, we have $B_1 - A = \binom{n}{p+2} T_s - \binom{n}{p} T_1 \geq 0$.

Similarly as the proof of Theorem 12, we can generate all the possible $a_1, a_2, a_3, \ldots, a_s$ from $p, p, \ldots, p, p+1, p+1, \ldots, p+1$ step by step. That is at each step we decrease some a_i by one and increase another a_j by one. Similarly as above, we can prove that the value of $\sum_{j=1}^{s}\sum_{i=0}^{a_j}\binom{n}{i}T_j$ will increase, after we doing such minor adjustment to these a_i. Therefore, we can conclude that all $\sum_{j=1}^{s}\sum_{i=0}^{a_j}\binom{n}{i}T_j$ are not smaller than A.

Now let's prove inequality (7). Based on (6), it is sufficient to show that $\sum_{i=0}^{u}\binom{n}{i}T_s > \sum_{j=1}^{s-1}\binom{n}{0}T_j + \sum_{i=0}^{u-s+1}\binom{n}{i}T_s$. Since we have

$$T_j/T_s \leq T_1/T_s \leq \binom{n}{\lceil (u-s+1)/s\rceil+1}/\binom{n}{\lfloor (u-s+1)/s\rfloor} < \binom{n}{u-s+1+j}/\binom{n}{0}$$

it means $\sum_{i=0}^{u}\binom{n}{i}T_s - \sum_{i=0}^{u-s+1}\binom{n}{i}T_s = \sum_{i=u-s+2}^{u}\binom{n}{i}T_s > \sum_{j=1}^{s-1}\binom{n}{0}T_j$. □

Based on the above theoretical results, similarly as algorithm $\mathbf{ISBS}_2$, we can implement an algorithm by using the s-direction strategy. Completely presenting this algorithm step by step is too complicated, thus here we only present the sketch of this algorithm:

1. Given an input system $\mathbb{P} : \{f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}, \ldots, f_{(s-1)n+1}, \ldots, f_{sn}\}$.
2. Generate s systems
   $\mathbb{P}_1 : \{f_1, \ldots, f_n, f_{n+1}, , \ldots, f_{2n}, \ldots, f_{(s-1)n+1}, \ldots, f_{sn}\}$
   $\mathbb{P}_2 : \{f_{n+1}, \ldots, f_{2n}, f_1, \ldots, f_n, f_{2n+1}, f_{2n+1}, \ldots, f_{3n}, \ldots, f_{(s-1)n+1}, \ldots, f_{sn}\}$
   $\vdots$
   $\mathbb{P}_s : \{f_{(s-1)n+1}, \ldots, f_{sn}, f_1, \ldots, f_n, f_{n+1}, \ldots, f_{2n}, \ldots, f_{(s-2)n+1}, \ldots, f_{(s-1)n}\}$
3. Set a total noise-bound u. Suppose $u-s+1 \equiv r \bmod s$. Let $p = (u-s+1-r)/s$.
4. For $\mathbb{P}_1, \ldots, \mathbb{P}_{s-r}$, set their partial noise-bounds to be p. For $\mathbb{P}_{s-r+1}, \ldots, \mathbb{P}_s$, set their partial noise-bounds to be $p+1$.
5. For i from 1 to s, solve $\mathbb{P}_i$ with **ISBS** method under total noise-bound u and its partial noise-bound minus 1.

- If for some $\mathbb{P}_k$, we achieve a solution x, set x_0 to be x and u to be $w(x)-1$, where $w(x)$ is the noise weight of x. Change the partial noise-bounds of $\mathbb{P}_{k+1}, \mathbb{P}_{k+2}, \ldots, \mathbb{P}_s$ by the strategy used in Step 4.

6. If x_0 is not empty, then its the solution we need. Otherwise, increase u by a step size, and repeat Step 4-5.

Remark 6. If the conditions of Theorem 17 are satisfied, the partial noise-bounds in Step 4 are optimal. Otherwise, they are good ones not the optimal ones.

8 Experimental Results

In order to test our improvements of **ISBS**, we generated some benchmarks from the Cold Boot key recovery problem of Serpent. The Cold Boot key recovery

problem originated from the Cold Boot attack, which was first proposed and discussed in the seminal work of [11]. The Cold Boot attack relies on the data remanence property of DRAM to retrieve memory contents after power off. In the Cold Boot attack to a block cipher, the attacker is able to retrieve the round keys, but some bits of the round keys is flipped since the decay of the memory data. Thus, the Cold Boot key recovery problem for a block cipher is to recover the initial key for these decayed round keys.

In [1], Cid and Albrecht proposed a mathematical model, by which they can convert the Cold Boot problem into the Partial Weighted Max-PoSSo problem, and they solved the Cold Boot key recovery problem of AES and Serpent by mixed integer programming solver **SCIP**. With this model, we used the original **ISBS** method to solve the same problems and achieved some better experimental results[15].

Since in this paper we only focus on solving the Max-PoSSo problem, unlike the general cold boot key recovery model, in our experiments we should assume that the bit decay in DRAM is symmetric: bit flips $0 \to 1$ and $1 \to 0$ occur with same probabilities δ. Under this assumption, the Cold Boot key recovery problem of a block cipher can be described as follows.

Let $\mathcal{KS} : \mathbb{F}_2^n \to \mathbb{F}_2^N$ be the key schedule function of a block cipher. where $N > n$. Let $G_{\mathcal{K}}$ be an equation system corresponding to $\mathcal{KS}$ such that the only pairs (k, K) that satisfy $G_{\mathcal{K}}$ are any $k \in \mathbb{F}_2^n$ and $K = \mathcal{KS}(k)$. Moreover, each $g_i \in G_{\mathcal{K}}$ has the form $h_i + K_i$ where h_i is some polynomial and K_i is the i-th bit of K. Let K' be the decayed round keys, and set $f_i = h_i + K'_i$ for any $1 \leq i \leq N$. Then, $\mathbb{P} = \{f_1, f_2, \ldots, f_N\}$ is the input of the Max-PoSSo problem we need to solve.

In [1] and [15], benchmarks with symmetric noise generated from the 128-bit versions of Serpent were tested, thus in the experiments of this paper, we solved the same benchmarks. Since we only use 256-bit round keys which means $m = 2n$, the improved **ISBS** algorithm we test is $\mathbf{ISBS}_2$ proposed in Section 7.1. The modified **ISBS** algorithm used in [15] is denoted by $\mathbf{ISBS}_m$ in the table, and **SCIP** correspond to the mixed integer programing method in [1].

We remind the reader that as mentioned in [15] when solving these benchmarks by $\mathbf{ISBS}_m$, the artificial bound strategy and the two-direction strategy were used, but the values of partial noise-bounds is not decided by `PartialBound`. Comparing with $\mathbf{ISBS}_m$, the main improvement of $\mathbf{ISBS}_2$ is that

- We apply the modification mentioned in Remark 1.
- We apply the order proposed in Section 3.3 when executing the incrementally solving process
- We compute the optimal partial noise-bound by `PartialBound` when using the two-direction strategy.

Our experimental platform is a PC with i7 2.8Ghz CPU(only one core is used), and 4G Memory, which is same as the one used in [15]. In our experiments, for each δ we generated 100 instances with random initial keys and random noise. As in [1, 15], we interrupted the solver when the running time exceeded the time limit 3600 seconds.

In the following table, the column "r" gives the success rate, which is the percentage of the instances we recovered the correct initial key, while the values in the brackets are the percentage of the instances we achieved the optimal solution within the time limit. Note that, there are two cases in which we cannot recover the correct initial key.

1. The solver was interrupted after the time limit.
2. The optimal solution achieved from the Max-PoSSo problems is not the true solution.

The column "avg. time" gives the average running time of the instances which are solved within the time limit, and the column "max t" gives the maximal running time for the instances which are solved with the time limit.

Table 3. Serpent considering $32 \cdot N$ bits of key schedule output (symmetric noise)

$\delta_0 = \delta_1$	Method	N	limit t	r	min t	avg. t	max t
0.01	**ISBS**$_2$	8	3600.0 s	100%	0.60 s	2.46 s	30.62 s
	ISBS$_m$	8	3600.0 s	100%	0.78 s	9.87 s	138.19 s
	SCIP	12	3600.0 s	96%	4.60 s	256.46 s	-
0.02	**ISBS**$_2$	8	3600.0 s	96(99)%	0.82 s	55.67 s	996.65 s
	ISBS$_m$	8	3600.0 s	96(99)%	0.80 s	163.56 s	2001.59 s
	SCIP	12	3600.0 s	79%	8.20 s	1139.72 s	-
0.03	**ISBS**$_2$	8	3600.0 s	91(95)%	0.58 s	171.17 s	2138.77 s
	ISBS$_m$	8	3600.0 s	90(92)%	1.74 s	314.78 s	3463.00 s
	SCIP	12	7200.0 s	53%	24.57 s	4205.34 s	-
0.05	**ISBS**$_2$	8	3600.0 s	40(98)%	3.67 s	382.61 s	1916.91 s
	ISBS$_m$	8	3600.0 s	38(94)%	12.37 s	745.80 s	2993.81 s
	SCIP	12	3600.0 s	18%	5.84 s	1921.89 s	-

From the experimental results, we can see that when $\delta = 0.01$, **ISBS**$_2$ is about 4 times faster than **ISBS**$_m$. When $\delta = 0.02, 0.03, 0.05$, **ISBS**$_2$ is about 2 times faster than **ISBS**$_m$.When $\delta = 0.05$, as in [15], we interrupted the solver after we have searched all the possible noise vectors under the noise-bound 12, thus although 98 instances returned the optimal solutions, only 40 of them were the true solutions. In summary, these experimental results show that with our modification we significantly improve the efficiency of **ISBS**.

9 The Comparison of the Algorithms for Solving Max-PoSSo Problems

In the precious sections, we introduced several algorithms for solving Max-PoSSo problems. They all based on the idea of exhaustive searching all the possible noise. In the three algorithms proposed above, **ISBS** is the basic algorithm. The

number of external paths of **ISBS** can reach its maximal value 2^n when the input system is totally random. In this case, the complexity of **ISBS** is $O(C_a \cdot 2^n)$, where C_a is the average complexity of solving a system corresponding to an external path in the search tree. When the input system is totally random and u_{opt}, the noise weight of the optimal solution, is n, the complexities of $\mathbf{ISBS}_b$ and $\mathbf{ISBS}_2$ can also reach this value. But when the input system satisfies some assumptions, $\mathbf{ISBS}_b$ and $\mathbf{ISBS}_2$ may have lower complexity. As we mentioned in Remark 3, the complexity of $\mathbf{ISBS}_b$ is lower than that of **ISBS** when u_{opt} is small. From the results in Section 7.1, we know that the complexity of $\mathbf{ISBS}_2$ is always lower than that of $\mathbf{ISBS}_b$ when $u_{opt} \leq n/2$. In general, $\mathbf{ISBS}_b$ and $\mathbf{ISBS}_2$ are originated from **ISBS** by adding some techniques to reduce the size of the searching tree. When these techniques are valid, the size of the search tree will be smaller than 2^n, hence $\mathbf{ISBS}_b$ and $\mathbf{ISBS}_2$ have lower complexity. However, there are a worst case in which these techniques are invalid and the size of the searching tree reachs its maximal value 2^n. This means that the upper bounds of the asymptotic complexities of these algorithms are almost the same.

Now we compare the class of **ISBS** algorithms with the algorithms based on the idea of searching the values of variables. A trivial algorithm based on this idea is evaluating $f_1(x), f_2(x), \ldots, f_m(x)$ at all the 2^n points in $\mathbb{F}_2^n$, and outputing the point x_0 such that the Hamming weight of $(f_1(x_0), f_2(x_0), \ldots, f_m(x_0))$ is minimal. The complexity of this trivial algorithm is $O(cm \cdot 2^n)$, where c is the complexity of evaluating one f_i at a point x in $\mathbb{F}_2^n$. Obviously, in most time cm is much less than C_a since evaluating is much easier than solving. Hence, in the worst cases, the complexity of the class of **ISBS** algorithms is worse than that of this trivial algorithm. However, a disadvantage for this trivial algorithm is that when the input system have some special properties, its complexity will not change. Therefore, when the randomness of the input system is not so good or u_{opt} is small, the class of **ISBS** algorithms is much more efficient. For example, for a system generated from Serpent in last section, which has 128 variables and 256 polynomials, evaluating the system 2^{16} times costs $26.2 \approx 2^{4.7}$ seconds by our implementation on the platform mentioned in last section, thus it will take about $2^{116.7}$ seconds to search all 2^{128} possible values of variables[2]. By comparing to the results in Table 8, we can see that $\mathbf{ISBS}_2$ is much more efficient when $\delta \leq 0.05$. In summery, when considering the upper bound of complexity, this trivial algorithm is slightly better than the class of **ISBS** algorithms, but from the viewpoint of practically computation, the class of **ISBS** algorithms is much better in most cases.

To the best of the author's knowledge, except the class of **ISBS** algorithms, other algorithms for solving Max-PoSSo problems, such as the Max-SAT and **MIP** solver, are all based on the idea of this trivial algorithm, and have different techniques to reduce the size of the searching space. For example, efficient **MIP** solver use a branch-and-cut algorithm as one of their core components, which

[2] Our implementation of evaluating may be not optimal, but even by an implementation which is 2^{10} times faster, it takes more than 2^{100} seconds to evaluate all 2^{128} possible values.

relaxes the problem to a floating point linear programming problem in order to prune search branches. As mentioned before, there is always a worst cases, in which these techniques are invalid, hence the size of the searching space is still 2^n. It means that the asymptotic complexities of these algorithms can be written as $O(C \cdot 2^n)$, where C is a non-exponential term when the size of input system is not exponential. We can see that except the differences of the non-exponential terms, the asymptotic complexities of these improved algorithms and the above trivial algorithm are the same. Hence, we think this trivial algorithm is almost the best algorithm for solving Max-PoSSo problems from the viewpoint of asymptotic complexity. For other algorithms based on the idea of searching the values of variables, they may have better performances than this trivial algorithm when the input system have some special properties. Since the specific complexities of these algorithm are unknown, we can only compare them with the class of **ISBS** algorithms by experiments. For example, in last section, we compared **SCIP**, which is one of these algorithms, with $\mathbf{ISBS}_2$ by solving the benchmarks generated from the problem of attacking Serpent.

Actually, a similar phenomenon occurs in the problem of solving Boolean polynomial system. To the best of the author's knowledge, for solving this problem, the algorithm with the best asymptotic complexity bound is the fast exhaustive search algorithm whose complexity is $O(dm \cdot 2^n)$, where d is the degree of the polynomial system [3]. Compared with the complexity of the trivial exhaustive search algorithm, this value only has a smaller non-exponential term. For other symbolic computation algorithms, such as the Gröbner basis method, the XL algorithm, and the characteristic set method, although they have good performances on solving some specific systems, their asymptotic complexity for solving general systems are all higher than that of the trivial exhaustive search algorithm. Hence, for solving PoSSo and Max-PoSSo problems over $\mathbb{F}_2$, an interesting open problem is whether there is a deterministic algorithm whose complexity is lower than $O(2^n)$ in any cases.

10 Conclusions

In this paper, we revisit the Max-PoSSo problem and the **ISBS** method. For the basic of Max-PoSSo, we show some results about the behavior of the success rate of recovering the true solution. For **ISBS**, we present several complexity bounds of it and propose some improvements in the general cases and overdetermined cases. We implement a new algorithm based on these improvements and test it by the Cold Boot Key recovery problem of Serpent with symmetric noise. The experimental results show that compared with the modified **ISBS** in [15], the new algorithm is about 2-4 times faster for different benchmarks.

There are two ideas of further improving **ISBS** which can be applied in the future. First, in our algorithm $\mathbf{ISBS}_b$ and $\mathbf{ISBS}_2$, when the noise-bound are gradually increased there are some repeated computations can be avoid. Actually, if a path is pruned because of the contradiction of polynomials, this path will also be pruned after the artificial noise-bound is increased. In other word,

after we increase the artificial noise-bound we only need to continue searching the paths which are pruned because of the former noise-bound. Therefore, if we can store the information of all these paths, we will reduce a lot of repeated computations. The second idea is that, for the two-direction or s-direction strategies, if we can find a better polynomial order such that incrementally solving the ordered system from all directions is more efficient, then we can improve the efficiency of the algorithm.

References

1. Albrecht, M.R. and Cid, C., Cold Boot Key Recovery by Solving Polynomial Systems with Noise. ACNS 2011: 57-72
2. Braeken A, Preneel B., Probabilistic algebraic attacks, Cryptography and Coding. Springer Berlin Heidelberg, 2005: 290-303.
3. Bouillaguet, C., Chen, H.-C., Cheng, C.-M., Chou, T., Niederhagen, R., Shamir, A., Yang, B.-Y.: Fast Exhaustive Search for Polynomial Systems in $\mathbb{F}_2$. CHES 2010. LNCS, vol. 6225, pp. 203-218. Springer, Heidelberg.
4. Chai, F., Gao, X.S. and Yuan, C., A Characteristic Set Method for Solving Boolean Equations and Applications in Cryptanalysis of Stream Ciphers, *Journal of Systems Science and Complexity*, 21(2), 191-208, 2008.
5. Courtois, N. and Pieprzyk, J., Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267-287. Springer, Heidelberg, 2002.
6. Cox, David A., *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra.* Springer, 2007.
7. Faugère, J.C., A New Efficient Algorithm for Computing Gröbner Bases (F4), *Journal of Pure and Applied Algebra*, 139(1–3), 61–88, 1999.
8. Faugère, J.C., A New Efficient Algorithm for Computing Gröner Bases Without Reduction to Zero (F5). In: Proc. ISSAC 2002. 75-83.
9. Gao, X.S. and Huang, Z., Characteristic set algorithms for equation solving in finite fields, Journal of Symbolic Computation, 47(6), 655-679, 2012.
10. Håstad, J., Satisfying Degree-d Equations over GF[2]^n. APPROX-RANDOM 2011: 242-253
11. Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul W., Calandrino, J. A., Feldman, A. J., Appelbaum, J. and Felten, E. W., Lest We Remember: Cold Boot Attacks on Encryption Keys. IN: USENIX Security Symposium, USENIX Association, pp. 45-60(2009).
12. Hartley, H. O., and E. R. Fitch., A chart for the incomplete beta-function and the cumulative binomial distribution. *Biometrika* 38.3/4 (1951): 423-426.
13. Huang, Z. and Lin, D. Attacking Bivium and Trivium with the Characteristic Set Method. Progress in Cryptology- AFRICACRYPT 2011, LNCS, Volume 6737, 77-91, 2011.
14. Huang, Z., Parametric Equation Solving and Quantifier Elimination in Finite Fields with the Characteristic Set Method. *Journal of Systems Science and Complexity*, 25(4), 778-791, 2012.
15. Huang, Z. and Lin, D., A New Method for Solving Polynomial Systems with Noise over $\mathbb{F}_2$ and Its Applications in Cold Boot Key Recovery, Selected Areas in Cryptography, pp. 16-33, LNCS 7707, Windsor, Canada, 2013.

16. May A., Meurer A., Thomae E., Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$. ASIACRYPT 2011: 107-124
17. Minto, S., Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *Proc. ACM/IEEE Design Automation*, 272-277, ACM Press, 1993.
18. Kamal, A.A., Youssef, A.M., Applications of SAT Solvers to AES key Recovery from Decayed Key Schedule Images. In: Proceedings of The Fourth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2010, Venice/Mestre, Italy, July 18-25 (2010).
19. Wu, W.T., Basic Principles of Mechanical Theorem-proving in Elementary Geometries, *Journal Automated Reasoning*, 2, 221-252, 1986.
20. Zhao, S.W. and Gao, X.S., Minimal Achievable Approximation Ratio for MAX-MQ in Finite Fields, *Theoretical Computer Science*, 410(21-23), 2285-2290, 2009.

Appendix: The strict proof of Proposition 2

In the following, we assume $1/r$ is an integer. First, we need the following two lemmas by which we can present the binomial cumulative function by the incomplete beta function. The two lemmas can be easily proved by integration by parts.

Lemma 18 *[12] Let n,c be positive integer, and $0 \leq p \leq 1$ is a real number. We have*

$$\sum_{k=0}^{c} \binom{n}{k} p^k (1-p)^{n-k} = \frac{\int_0^{1-p} t^{n-c-1}(1-t)^2 dt}{B(n-c, c+1)},$$

where $B(a,b)$ is the Beta function.

Lemma 19 *[12] Let $I_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}dt}{B(a,b)}$. This $I_x(a,b)$ is called the incomplete beta function. Then, we have*

(1) $I_x(a,b) = 1 - I_{1-x}(b,a)$

(2) $I_x(a+1,b) = I_x(a,b) - \frac{x^a(1-x)^b}{aB(a,b)}$.

Proof of Proposition 2:

$$\sum_{i=0}^{a}\left(\binom{n-k}{i}\sum_{j=0}^{a+b-i}\binom{k}{j}\right) + \sum_{i=0}^{b}\binom{n}{i} \geq \sum_{i=0}^{a}\binom{n}{i} + \sum_{i=0}^{b}\binom{n}{i}$$

.

Let $k = mr$, and $s = 1/r$ is an integer. From the above two lemmas, we have $f(k) = \frac{\sum_{i=0}^{k}\binom{m}{i}}{2^m} = I_{1/2}(k(s-1), k+1)$. Then, the conclusion is equivalent to $f(k+t) \leq f(k)$ for any $k, t \in N$. Thus, it is sufficient to prove $f(k+1) \leq f(k)$, $\forall k \in N$.

Note that by Lemma 19, we have

$$f(k+1) = I_{1/2}((k+1)(s-1), k+2) = 1 - I_{1/2}(k+2, (k+1)(s-1))$$
$$= 1 - I_{1/2}(k+1, (k+1)(s-1)) + \frac{1}{2^{(k+1)s}(k+1)B(k+1, (k+1)(s-1))}$$
$$= I_{1/2}((k+1)(s-1), k+1) + \frac{1}{2^{(k+1)s}(k+1)B(k+1, (k+1)(s-1))}$$

For simplicity, we set $a = (k+1)(s-1)$, $b = k+1$. By applying (2) of Lemma 19 $s-1$ times, we have

$$I_{1/2}(a,b) = I_{1/2}(a-(s-1), b) - \sum_{i=1}^{s-1} \frac{1}{2^{a+b-i}(a-i)B(a-i,b)}.$$

Note that $I_{1/2}(a-(s-1), b) = I_{1/2}(k(s-1), k+1) = f(k)$. Thus,

$$f(k+1) - f(k) = \frac{1}{2^{a+b}bB(b,a)} - \sum_{i=1}^{s-1} \frac{1}{2^{a+b-i}(a-i)B(a-i,b)} \tag{8}$$

Since $1 \le i \le s$, we have

$$\frac{1}{(a-i)B(a-i,b)} = \frac{1}{(a+b-i)B(a-i+1,b)}$$
$$= \frac{a-i+1}{(a+b-i)(a+b-i+1)B(a-i+2,b)}$$
$$= \cdots = \frac{(a-i+1)(a-i+2)\cdots(a-1)}{(a+b-i)(a+b-i+1)\cdots(a+b-1)B(a,b)}$$
$$\ge \frac{(a-s+1)^{i-1}}{(a+b-1)^i B(a,b)}$$

Therefore, by applying the above inequality to (8), we have

$$f(k+1) - f(k) \le \frac{1}{2^{a+b}bB(b,a)} - \frac{1}{2^{a+b}(a-s+1)B(a,b)} \sum_{i=1}^{s-1} (\frac{2(a-s+1)}{a+b-1})^i.$$

Let $q = \frac{2(a-s+1)}{a+b-1}$, then $f(k+1) - f(k) = \frac{1}{2^{a+b}B(b,a)}(\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}))$.

Now, it is sufficient to show $\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}) \le 0$. If $s = 2$, the conclusion is correct obviously. Now we consider the case $s \ge 3$. In this case, $q - 1 = \frac{2(a-s+1)}{a+b-1} = \frac{(k-1)(s-2)-1}{a+b-1} > 0$. Thus, $\frac{1-q^{s-1}}{1-q} = 1 + q + \cdots + q^{s-2} > s-1$. Then $\frac{1}{b} - \frac{2}{a+b-1}(\frac{1-q^{s-1}}{1-q}) < \frac{1}{k+1} - \frac{2(s-1)}{(k+1)s-1} < \frac{1}{k+1} - \frac{2(s-1)}{(k+1)s} < \frac{2-s}{(k+1)s} < 0$. □