The Implementation and Complexity Analysis of the Branch Gröbner Bases Algorithm Over Boolean Polynomial Rings

Yao Sun and Dingkang Wang

Abstract A new branch of Gröbner basis algorithm over boolean ring has been presented in an earlier paper. In this paper, the detailed implementation and a rough complexity analysis is given. The branch Gröbner basis algorithm implements a variation of the F5 algorithm and bases on the ZDD data structure, which is also the data structure of the framework PolyBoRi. This branch Gröbner basis algorithm is mainly used to solve algebraic systems and attack multivariable cryptosystems, and its goal is to lower the complexity in each branch and expect better total complexity. An important proposition ensures the two original criteria of the non-branch F5 algorithm could still reject almost all unnecessary computations in this new branch algorithm. The timings show this branch algorithm performs very well for randomly generated systems as well as a class of stream ciphers which is generated by the linear feedback shift register (LFSR).

1 Introduction

Solving system of polynomial equations is a basic problem in computer algebra, through which many practical problems can be solved easily. Among all the methods for this purpose, Gröbner bases method, the characteristic set method and resultant method are the most famous ones [1, 2].

Since Buchberger proposed the Gröbner bases algorithm in 1965, this algorithm has been improved by many researchers, both from the data structure and the criteria to remove the redundant S-pairs. Now the most famous Gröbner bases algorithms are the F4 and F5 algorithms proposed by Faugère [3, 4]. The F4 algorithm imports

Y. Sun (✉)
SKLOIS, Institute of Information Engineering, CAS, 100093 Beijing, China
e-mail: sunyao@iie.ac.cn

D. Wang
KLMM, Academy of Mathematics and Systems Science, CAS,
100190 Beijing, China
e-mail: dwang@mmrc.iss.ac.cn

R. Feng et al. (eds.), *Computer Mathematics*, DOI 10.1007/978-3-662-43799-5_14

the matrix technique to make the reduction process more efficient, while the F5 algorithm presents two new criteria to eliminate the useless S-pairs, which can be definitely reduced to 0.

So far, both F4 and F5 algorithms have been implemented. The most efficient implementation of F4 algorithm is presented by Steel, and is available on the computer algebraic system Magma, while the the most efficient version of F5 algorithm is implemented by Faugère himself, which is not open. However, the F4 algorithm is still not perfect, as high efficiency leads to the cost of enormous memories. For example, attacking the cryptographic system HFE80 by using F4 algorithm in Magma will cost nearly 16G memories.

For solving system of boolean polynomial equations, a Gröbner bases algorithm based on the ZDD data structure has been proposed by Brickenstein in 2007 (the PolyBoRi framework) [5]. His algorithm works very well for computing Gröbner basis with the pure lexicographic monomial order. However, since it is expensive to compute the total degree leading monomial for a polynomial in the ZDD form, this algorithm possibly does not perform very well with total degree orders. So in our implementation, we prefer a new graded expression of polynomials such that the leading monomial for total degree order can be calculated extremely fast, so our algorithm is more efficient with graded monomial orders. A characteristic set method for solving system of boolean polynomial equations is presented by Gao, and his method has pretty good performance on the problem of stream cipher systems [6]. Gao's implementation is also based on ZDD data structure and he uses the branch technique to compute the ascending set series.

The success of Gao's algorithm is a motivation for our research on branch Gröbner basis. Our algorithm makes improvements both on saving the usage of memories and limiting the size of matrices. That is, on one hand, we utilize the ZDD data structure to save polynomials and decrease the cost of space, and on the other hand, we make new branches when the matrix grows bigger so that we only need to handle matrix with a reasonable size.

In theory, we employ a modified matrix F5 algorithm. Details can be found in [7]. In this paper, we concentrate on the implementation and complexity analysis of our algorithm. The contents of this paper are organized as follows: the second section involves some preliminaries and the modified algorithm; the third section introduces the ZDD data structure and some sub-algorithms; complexity analysis comes in the fourth section; some examples and timings are given in the fifth section; we end this paper with conclusions.

2 The Algorithm

2.1 Notations

Let F_2 be the finite field with two elements 0 and 1, and $X = x_1, \ldots, x_n$ stands for the set of variables. Let H be the set of field polynomials $\{x_1^2 + x_1, \ldots, x_n^2 + x_n\}$, then

the ring $R_2 = F_2[X]/\langle H\rangle$ is actually a boolean ring, where $\langle H\rangle$ is the ideal generated by H in $F_2[X]$, and in addition, we call the elements in R_2 boolean polynomials.

Let N be the set of nonnegative integer and T be the power set of X, which means $T = \{x_1^{\alpha_1}\cdots x_n^{\alpha_n} | \alpha_i \in \{0, 1\}, i = 1, \ldots, n\}$. Assume $\prec$ is an admissible monomial order defined over T, then given $t = x_1^{\alpha_1}\cdots x_n^{\alpha_n} \in T$, we define the degree of t as $\deg(t) = \sum_{i=1}^{n} \alpha_i$. For a polynomial $0 \neq f \in F_2[X]$, we have $f = \sum x_1^{\alpha_1}\cdots x_n^{\alpha_n}$. Define the degree of f as $\deg(f) = \max\{\alpha_1 + \cdots + \alpha_n\}$, while the leading monomial of f is $\mathrm{lm}(f) = \max_{\prec}\{x_1^{\alpha_1}\cdots x_n^{\alpha_n}\}$.

2.2 Definitions

To introduce our algorithm, some definitions are necessary and the following definitions are imported from [7]:

Definition 1 Let $L = T \times F_2[X] \times \mathrm{N} \times F_2[X] \times \mathrm{N}$, and a labeled polynomial is a five-tuple vector $\mathscr{G} = (x^\alpha, f, i, g, k) \in L$. We define the signature of $\mathscr{G}$ as $\mathrm{S}(\mathscr{G}) = x^\alpha$, the initial as $\mathrm{init}(\mathscr{G}) = f$, the extended signature as $\mathrm{ES}(\mathscr{G}) = \mathrm{S}(\mathscr{G})\mathrm{lm}(\mathrm{init}(\mathscr{G})) = x^\alpha \mathrm{lm}(f)$, the index as $\mathscr{G}) = i$, the polynomial as $\mathrm{poly}(\mathscr{G}) = g$ and the number as $\mathrm{num}(\mathscr{G}) = k$.

Definition 2 Let $\mathscr{F}, \mathscr{G} \in L$ be two labeled polynomials. We say $\mathscr{F} \prec_{es} \mathscr{G}$ (or $\mathscr{G} \succ_{es} \mathscr{F}$), if one of the following three cases is satisfied: 1. $\mathrm{ES}(\mathscr{F}) \prec \mathrm{ES}(\mathscr{G})$. 2. $\mathrm{ES}(\mathscr{F}) = \mathrm{ES}(\mathscr{G})$ and $\mathscr{F}) > \mathscr{G})$. 3. $\mathrm{ES}(\mathscr{F}) = \mathrm{ES}(\mathscr{G})$, $\mathscr{F}) = \mathscr{G})$ and $\mathrm{num}(\mathscr{F}) > \mathrm{num}(\mathscr{G})$.

The order of labeled polynomials, which alleviate the influence of the order of the input polynomials or the initial polynomials, is modified from the F5 algorithm. Therefore, the criteria should be revised correspondingly. In the following two definitions, let $\mathscr{F} \in L$ be a labeled polynomial and $B \subset L$ be a set of labeled polynomials.

Definition 3 We say $\mathscr{F}$ is normalized by B, if there do not exist a labeled polynomial $\mathscr{G} \in B$ and a monomial $v \in T$, such that $\mathrm{S}(\mathscr{F}) = v\mathrm{lm}(\mathrm{poly}(\mathscr{G}))$, $\mathscr{F} \succ_{es} v\mathrm{lm}(\mathrm{init}(\mathscr{F}))\mathscr{G}$. Particularly, in the boolean ring R_2, the condition $\mathrm{lm}(\mathrm{init}(\mathscr{F})) \nmid \mathrm{S}(\mathscr{F})$ should hold as well.

Definition 4 We say $\mathscr{F}$ can be rewritten by B, if there exists a labeled polynomial $\mathscr{G} \in B$ and a monomial $v \in T$, such that $\mathrm{S}(\mathscr{F}) = \mathrm{S}(v\mathscr{G})$, $(\mathscr{F}) = (\mathscr{G})$ and $\mathrm{num}(\mathscr{F}) < \mathrm{num}(\mathscr{G})$.

2.3 Criteria

Now, we give two new criteria modified from the F5 algorithm without proofs. A partial proof can be found in [7], and the complete one will come in a future paper.

1. **Syzygy criterion:** Given a critical pair: $s(\mathscr{G}_1, \mathscr{G}_2) = (m, u_1, \mathscr{G}_1, u_2, \mathscr{G}_2)$, where $u_1, u_2 \in T$ and $\mathscr{G}_1, \mathscr{G}_2 \in B$. If either $u_1\mathscr{G}_1$ or $u_2\mathscr{G}_2$ is not normalized by B, then the critical pair $s(\mathscr{G}_1, \mathscr{G}_2)$ can be discarded.
2. **Rewritten criterion:** Given a critical pair: $s(\mathscr{G}_1, \mathscr{G}_2) = (m, u_1, \mathscr{G}_1, u_2, \mathscr{G}_2)$, where $u_1, u_2 \in T$ and $\mathscr{G}_1, \mathscr{G}_2 \in B$. If either $u_1\mathscr{G}_1$ or $u_2\mathscr{G}_2$ can be rewritten by B, then the critical pair $s(\mathscr{G}_1, \mathscr{G}_2)$ can be discarded.

Our non-branch Gröbner bases algorithm is nothing else than a general Buchberger algorithm except replacing the polynomials with the labeled polynomials, adding two criteria and using matrix reduction. After revising the two criteria, our algorithm has less influence from the input order of the initial polynomials than the F5 algorithm. Furthermore, the two revised criteria can also eliminate almost all the useless critical pairs as the F5 algorithm does. For semi-regular systems, there does not exist the critical pairs that can be reduced to 0, too. In fact, we proved in [7] that the modifications in the comparison of two labeled polynomials do not affect the function of the criteria. To illustrate how the two new criteria work, we have tested some randomly generated boolean polynomial systems in Sect. 4.

2.4 Trick

Although the two modified criteria can remove almost all the useless critical pairs generated during the computation, the total efficiency is not as good as we wished. One possible reason may be the conflict between the signature and the polynomial.

In fact, the motivation of the signature is to record the origin of the present label polynomial. Take a labeled polynomial, say $\mathscr{G} = (x^\alpha, f, i, g, k) \in L$, for example. The signature tells us that the polynomial g is obtained by reducing the polynomial $x^\alpha f_i$ with 'smaller' labeled polynomials under the order $\prec_{es}$. So the signature actually works as a clue of the computation, and that is exactly why the two criteria work. However, there exists a natural conflict between the signature and the polynomial. That is, during the computation, the label polynomials generated later sometimes have smaller size but with bigger signature. By our algorithm as well as the F5 algorithm, the label polynomial with a bigger signature should be dealt with later. It is possible that some polynomials of smaller size cannot be used immediately and this may lead to more computations. One trick can be used to solve this conflict. The key idea is to clear the signatures of some simple polynomials and to append them to the initial polynomials. We have

Proposition 1 *Adding an initial polynomial with the largest index at any time will not affect the correctness of the two criteria.*

Although adding new polynomials will not affect the correctness of the two criteria, in order to keep the algorithm correct, we must add polynomials that are in the original ideal. Usually, we add new initial polynomials in the following cases:

1. The new generated polynomial has a very low total degree, such as 1 or 2.

2. The leading monomials of some present initial polynomials can be reduced by the new generated polynomial.

However, adding too many polynomials cannot speedup the algorithm. Because adding new initial polynomials is actually to cut off the relationship between this polynomial and the initial polynomials, which will weaken the criteria, since the system of initial polynomials may not be semi-regular any more, and many useless pairs cannot be detected.

2.5 Branch Strategy

In both the F4 and F5 algorithms, the sizes of matrices in the computation grow quickly with the degree of critical pairs. Huge matrices occupy enormous memories and are difficult to deal with. In consideration of the complexity, a natural idea comes to us, and that is, we should prevent the degree of critical pairs growing too high.

In order to control the size of the matrix, we can add polynomials that are not ideal, but this will apparently make the output incorrect. Fortunately, the cases are better in the boolean ring R_2. Since in R_2, any boolean polynomial has only two possible values 0 and 1, which makes the branch algorithm available. We can clone the present system and add a new polynomial with the value 0 and 1 to them respectively such that two polynomial sets are obtained. The original ideal is the intersection of the ideals generated by these two polynomial sets. This fact is important for solving the original system. Furthermore, after adding the new polynomial, each system may have smaller matrices and are easier to be dealt with. This is the motivation of our branch algorithm.

Theoretically, any polynomial can be added. In consideration of the complexity, we usually add polynomials that are simple enough or that can be used to reduce other polynomials. When should we add polynomials? Based on our experiments, we prefer to add polynomials when the degree of critical pairs is high enough. We can set a degree bound $D \in \mathrm{N}$. When the remaining pairs have higher degrees than D, we add a new polynomial, or equivalently we make a new branch.

We have many options to choose the new polynomials and it is difficult to tell which is the best. So we list some alternatives that have good performance in the experiments.

1. The polynomial with degree one.
2. The polynomial which is the highest homogeneous part of some present polynomial.
3. The polynomial with a high reference degree, which is a parameter that comes from the shared ZDD data structure.

After all, we can present our branch Gröbner bases algorithm.

Algorithm 1 : Branch Grönber bases algorithm

Input: An ordered polynomials set $F = (f_1, \ldots, f_m) \subset \mathrm{F}_2[X]$.
Output: The branch Gröbner bases *BranchGB* of the ideal generated by $F \cup H$, where $H = \{x_1^2 + x_1, \ldots, x_n^2 + x_n\} \subset \mathrm{F}_2[X]$ are the field polynomials.

1 Set $\mathscr{F}_i := (1, f_i, i, f_i, i)$, $i = 1, \ldots, m$, $\mathscr{F}_{m+i} := (1, x_i^2 + x_i, m + i, x_i^2 + x_i, m + i)$, $i = 1, \ldots, n$.
index $:= m + n$, $k := m + n$, *BranchSet* $:= \{\{\mathscr{F}_i | i = 1, \ldots, k\}\}$, *BranchGB* $:= \{\}$.
2 While *BranchSet* $\neq \emptyset$ do
2.1 Select a $B \in$ *BranchSet* and *BranchSet* $:=$ *BranchSet* $\setminus \{B\}$.
2.2 Generate $CP := \{\mathrm{s}(\mathscr{P}, \mathscr{Q}) | \mathscr{P}, \mathscr{Q} \in B\}$.
2.3 While $CP \neq \emptyset$ do
2.3.1 $d := \min\{\deg(c) | c \in CP\}$, $D := \{c \in CP | \deg(c) = d\}$ and $CP := CP \setminus D$.
2.3.2 $D' := \{c \in D | c$ is not satisfied either of the Syzygy or Rewritten criterion $\}$.
2.3.3 Reduce D' by matrix and collect the new generated labeled polynomials as F^+.
2.3.4 For $\mathscr{P} \in F^+$ do
2.3.4.1 If $\mathscr{P}$ is simple enough
then *index* $:=$ *index* $+ 1$, $\mathscr{P} := (1, \mathrm{poly}(\mathscr{P}),$ *index*, $\mathrm{poly}(\mathscr{P}), k + 1)$.
else $\mathrm{num}(\mathscr{P}) := k + 1$.
2.3.4.2 $k := k + 1$, $CP := CP \cup \{\mathrm{s}(\mathscr{P}, \mathscr{Q}) | \mathscr{Q} \in B\}$. $B := B \cup \{\mathscr{P}\}$.
2.3.5 If the minimal degree of CP is high enough, then
2.3.5.1 choose a new polynomial $p \in \mathrm{F}_2[X]$.
2.3.5.2 *index* $:=$ *index* $+ 1$, $k := k + 1$
2.3.5.3 Set $\mathscr{P}' := (1, p + 1,$ *index*, $p + 1, k)$, *BranchSet* $:=$ *BranchSet* $\cup \{B \cup \{\mathscr{P}'\}\}$.
2.3.5.4 Set $\mathscr{P} := (1, p,$ *index*, $p, k)$, $CP := CP \cup \{\mathrm{s}(\mathscr{P}, \mathscr{Q}) | \mathscr{Q} \in B\}$, $B := B \cup \{\mathscr{P}\}$.
2.4 *BranchGB* $:=$ *BranchGB* $\cup \{\{\mathrm{poly}(\mathscr{P}) | \mathscr{P} \in B\}\}$.
3 Return *BranchGB*.

In this algorithm, step 2.3.4.1 is to clear the signature of a polynomial, and step 2.3.5 is to introduce new polynomials so as to add new branches to the algorithm. In order to make the algorithm more efficient, some auxiliary data can be kept in *BranchSet* as well, such as the number *index*, k, the set CP and so on.

3 Complexity Analysis

3.1 The Principle for Making Branch

One of the motivations of our branch Gröbner bases algorithm is to decrease the complexity in each branch so as to lessen the total complexity for solving the boolean polynomial system. Therefore, if we hope our branch algorithm has better efficiency than the non-branch algorithms, we should constrain the branch number within a reasonable bound.

The complexity of Gröbner bases using the $F4$ and $F5$ algorithm is determined by a degree D_{reg}, which is the upper bound of the highest degree of matrices, say D_{p}, constructed during the computation. Then the total complexity of $F4$ or $F5$ algorithm is roughly $\mathrm{O}(n^{\omega D_{\mathrm{reg}}})$, where n is the number of variables and ω is the efficiency of

matrix elimination that has a bound $2 \leq \omega \leq 3$. Similarly, since we make a new branch when the degree of pairs exceed a degree D, the complexity of each branch is roughly $O(n^{\omega' D})$, where ω' is efficiency of our matrix reduction. Assuming the number of branches is M, the total complexity of our branch algorithm is $O(Mn^{\omega' D})$. If we hope our branch algorithm performs better, the complexity $O(Mn^{\omega' D})$ must smaller than $O(n^{\omega D_{\text{reg}}})$. Roughly speaking, the number M should satisfy the following inequality:

$$Mn^{\omega' D} < n^{\omega D_{\text{reg}}}, \quad \text{or} \quad M < n^{\omega D_{\text{reg}} - \omega' D}.$$

However, it is difficult to predict the number of branches produced in our algorithm. What we can do is to set up a bound such that when the number of branches exceed it, we stop the program. Fortunately, the number of branches for one kind of examples is usually stable, so we can anticipate the general performance for all problems of this kind by induction. The principle above sets up a criterion to check whether it is possible for our algorithm to have good performance.

3.2 The Estimation of D_{reg}

Before the system is computed by the $F5$ algorithm, we cannot obtain the practical value of D_{p}, so we have to try to estimate it or give an upper bound for it. Fortunately, the upper bound D_{reg} for F5 algorithm can be obtained from [8]. For a general semi-regular system in the boolean ring, D_{reg} can be achieved from the following series:

$$S_{m,n}(z) = \sum_{d \geq 0} h_{d,m}(n) z^d = (1+z)^n / \prod_{k=1}^{m} (1 + z^{d_k}).$$

where m is the number of the initial polynomials $(f_1, \ldots, f_m)$ and $d_k = \deg(f_k)$. Then D_{reg} is the first d such that $h_{d,m}$ is nonpositive. Therefore, the upper bound can be calculated easily when m, n, and the d_ks are given.

3.3 Theoretical Analysis of Randomly Generated Systems

Faugère has claimed that almost all the systems are semi-regular systems when n is not too small. So the randomly generated systems can be supposed to be semi-regular systems naturally, and the estimation of D_{reg} is a powerful tool to analyze the complexity of the F5 algorithm as well as our branch algorithm.

For convenience, we only consider some special cases and the others can be analyzed in a similar way. We assume the initial polynomials are m quadratic polynomials in R_2 and n is the number of variables. So when $m = n$, we can draw a picture of D_{reg}.

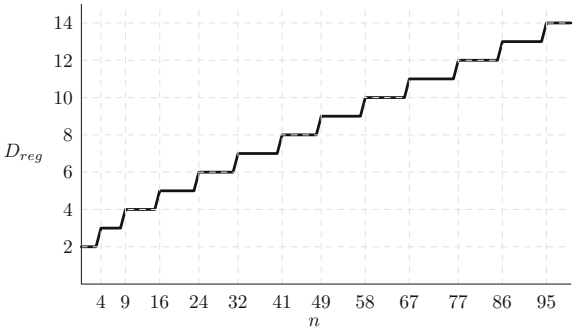
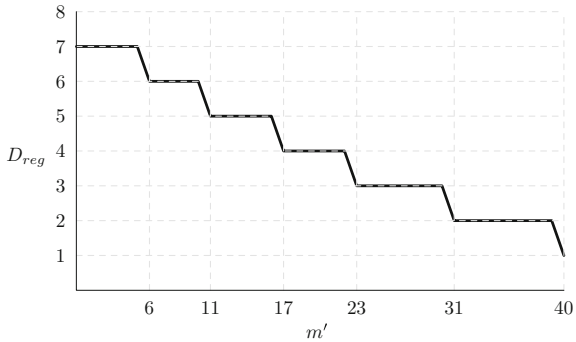


Fig. 1 D_{reg} for $m = n$ quadratic polynomials

Fig. 2 D_{reg} when adding polynomials with degree 1



Although D_{reg} is only an upper bound of the practical highest degree D_{p}, the degree D_{reg} is reached almost all the time for randomly generated systems.

The motivation of our branch algorithm is to add new polynomials in order to lower the degree of matrices in the computation. Since randomly generated systems are semi-regular, we can use the series in the last subsection to compute D_{reg} and then find out how many polynomials should be added such that D_{reg} becomes lower. If the number of polynomials is fixed, the number of branches can be obtained easily.

For example, we consider the systems when $m = n = 40$ and in Fig. 1, the corresponding D_{reg} is 7. By the series, we can calculate the specific number of polynomials we should add in order to lower the degree. Since adding different polynomials lead to different results, here we only consider two cases, one is adding polynomials with degree 1 and the other with degree 2. Figure 2 shows how the D_{reg} varies with m', which is the number of new added polynomials. We can see that adding 6 polynomials of degree 1 will lower D_{reg} to 6, while adding 11 polynomials, the D_{reg} becomes 5 and so on. Then we obtain the following table. TH-Num is the theoretic branch number calculated by the inequality in Sect. 3.1 with $\omega = \omega' = 2$. NumPoly is the smallest number of m' to lower D_{reg} to the corresponding degree and EXP-Num is the expected branch number calculated by NumPoly.

Table 1 Adding polynomials with degree 1

D_{reg}	6	5	4	3	2	1
TH-Num	1,600	40^4	40^6	40^8	40^{10}	40^{12}
NumPoly	6	11	17	23	31	40
EXP-Num	64	2,048	2^{17}	2^{23}	2^{31}	2^{40}
TH/EXP($\approx$)	$2^{4.6}$	$2^{10.3}$	$2^{14.9}$	$2^{19.6}$	$2^{22.2}$	$2^{23.9}$

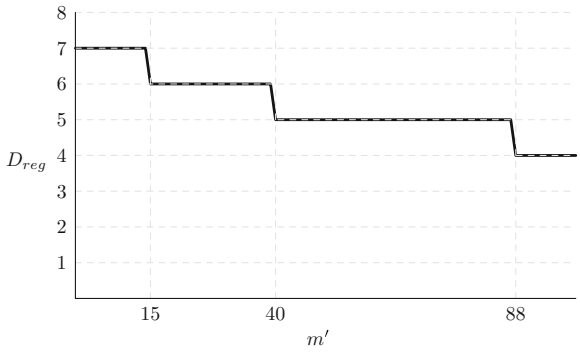


Fig. 3 D_{reg} when adding polynomials with degree 2

In Table 1, all the expected branch numbers are smaller than the theoretical bound, so as discussed in 3.1, our branch algorithm will have better performance than the non-branch algorithm and the practical results in Sect. 4 proves this. Recall that, the proportion in Table 1 shows that adding 40 polynomials will lead to the best efficiency, however, this is not the truth, since the matrices with lower degree are denser than the bigger ones, then the parameter ω' is no longer 2 and combined with other factors, TH-Num will be much smaller than that in the table.

When adding polynomials with degree 2, the cases are not so good as that with degree 1, and Fig. 3 shows the variation of D_{reg}. At least 15 quadratic polynomials should be added to lower D_{reg} to 6, while in Fig. 2, only 6 is enough, so the corresponding expected branch number will be bigger than in Table 1. We can generate a similar table (Table 2).

The expected branch number are all bigger than the theoretic bound, so we cannot expect a better performance of our branch algorithm. However, the practical cases are not as bad as the table shows, since the data in the table are calculated under a hypothesis that the new systems are still semi-regular systems, and so if we add some special polynomials which can reduce the present system significantly, the practical branch number can still be control in a reasonable bound.

Other examples can be analyzed in the same way as well. Based on our experimental results, we can conclude that for almost all randomly generated systems, our branch algorithm has better performance than non-branch algorithms, such as F4 in Magma.

Table 2 Adding polynomials with degree 2

D_{reg}	6	5	4	3	2
TH-Num	1,600	40^4	40^6	40^8	40^{10}
NumPoly	15	40	88	207	740
EXP-Num	2^{15}	2^{40}	2^{88}	2^{207}	2^{704}

Table 3 The criteria

n	6	8	10	12	14	16	18
T-pairs	198	740	2,483	3,296	94,077	144,801	211,385
D-pairs	7	64	727	1,146	46,221	80,308	112,925
0-Polys	0	0	0	0	0	0	0
D/T(%)	3.54	8.65	29.28	34.77	49.13	55.46	53.42

4 Computational Examples

In this section, some timings are imported from [7], but the complexity information is obtained by the method discussed in Sect. 3. The branch strategy we used in the experiments is the first strategy in Sect. 2.5 and all the timings are obtained from a computer (OS Linux, CPU Xion 4*3.0GHz, 16.0GB RAM).

4.1 The Criteria

In this subsection, we see how the two modified criteria work. The initial polynomials are all randomly generated quadratic polynomials with $m = n$.

In Table 3, T-pairs stand for the total number of pairs generated in the computation, while D-pairs are the number of pairs detected by the two criteria and D/T is the proportion of these two numbers. Besides, 0-Polys is the number of polynomials that are reduced to 0 in the algorithm. From the table, all useless pairs are detected and the two criteria play an important role for improving the efficiency during the computation.

4.2 The Randomly Generated Systems

In this section we see how the branch algorithm performs for randomly generated systems. In the following table, three groups of data will be presented. The first group involves the theoretical degree bound D_{reg} and the theoretical upper bound for the branch number M; the second group consists of the practical degree D_{p} computed by the F4 algorithm and the corresponding theoretical upper bound for M; in the

Table 4 The randomly generated systems

n	*EST*			*MGB*			*BGB*	
	TH-Deg	TH-Num	P-Deg	TH-Num	Time	P-Deg	P-Num	Time
18	5	18^4	6	18^6	3.890	3	2,048	0.791
20	5	20^4	6	20^6	14.220	3	8,160	2.992
22	5	22^4	6	22^6	82.790	3	29,046	13.235
24	6	24^6	–	–	–	3	65,400	47.682
26	6	26^6	–	–	–	3	262,018	149.121

last group, the degree D and the practical number of the branches is given. For the second and third groups, the practical timings are given as well.

The initial polynomials are randomly generated quadratic polynomials with $m = n$. The practical degree D_{p} in the second group are obtained by the F4 algorithm in Magma. The upper bound of M is estimated with $\omega = \omega' = 2$. In Table 4, we use *EST* to represent the estimated data and *MGB* is the experiments data from the F4 algorithm, while *BGB* is that from our branch algorithm. TH- is short for theoretical and P- for practical and "–" means Magma runs out of memory.

Remark that in Table 4, the theoretical degrees for $n = 18, 20, 22$ are lower than the corresponding practical degrees, and the reason is that the F4 algorithm does not have a powerful criteria to remove all the useless pairs, so some useless computations are still done in the F4 algorithm. From the table we can also see that the practical numbers of branches in our algorithm is always kept within the two theoretical bounds, so our branch algorithm will have better performance and the timings prove that.

4.3 The Stream Ciphers

In this section, we use our branch algorithm to attack a class of stream ciphers, which is an important class of encryption algorithm. Here we only consider stream ciphers based on the LFSR, and the filter functions are from [9].

- CanFil 2, $x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 + x_1x_4 + x_2x_5 + x_3 + x_4 + x_5$
- CanFil 3, $x_2x_3x_4x_5 + x_1x_2x_3 + x_2x_4 + x_3x_5 + x_4 + x_5$
- CanFil 8, $x_1x_2x_3 + x_2x_3x_6 + x_1x_2 + x_3x_4 + x_5x_6 + x_4 + x_5$
- CanFil 9, $x_2x_4x_5x_7 + x_2x_5x_6x_7 + x_3x_4x_6x_7 + x_1x_2x_4x_7 + x_1x_3x_4x_7 + x_1x_3x_6x_7 + x_1x_4x_5x_7 + x_1x_2x_5x_7 + x_1x_2x_6x_7 + x_1x_4x_6x_7 + x_3x_4x_5x_7 + x_2x_4x_6x_7 + x_3x_5x_6x_7 + x_1x_3x_5x_7 + x_1x_2x_3x_7 + x_3x_4x_5 + x_3x_4x_7 + x_3x_6x_7 + x_5x_6x_7 + x_2x_6x_7 + x_1x_4x_6 + x_1x_5x_7 + x_2x_4x_5 + x_2x_3x_7 + x_1x_2x_7 + x_1x_4x_5 + x_6x_7 + x_4x_6 + x_4x_7 + x_5x_7 + x_2x_5 + x_3x_4 + x_3x_5 + x_1x_4 + x_2x_7 + x_6 + x_5 + x_2 + x_1$
- CanFil 10, $x_1x_2x_3 + x_2x_3x_4 + x_2x_3x_5 + x_6x_7 + x_3 + x_2 + x_1$

The data here consist of two parts: one part involves the practical degree D_{p}, the theoretical upper bound for M and the practical time costed by the F4 algorithm in Magma; the other part includes the degree D which is given by the user, the practical branch number and the time used by our branch algorithm. Again, $\omega = \omega' = 2$

Table 5 The stream ciphers

Filters	n	81			100			128		
		Time	Deg	Num	Time	Deg	Num	Time	Deg	Num
	MGB	18.730	7	81^8	32.930	7	100^8	–	–	–
CanFil2	*BGB*	0.027	3	9	0.172	3	66	0.357	3	40
	MGB	–	–	–	1.360	7	100^8	–	–	–
CanFil3	*BGB*	0.085	3	11	0.150	3	19	1.210	3	17
	MGB	49.460	7	81^8	12.590	7	100^8	–	–	–
CanFil8	*BGB*	0.046	3	27	0.170	3	190	0.371	3	140
	MGB	–	–	–	–	–	–	–	–	–
CanFil9	*BGB*	0.418	4	40	1.230	4	217	20.901	4	77
	MGB	331.880	7	81^8	–	–	–	–	–	–
CanFil10	*BGB*	0.131	3	99	0.612	3	501	1.296	3	340

and *MGB* and *BGB* represent the F4 algorithm in Magma and our branch algorithm respectively.

Table 5 shows that the practical number of branches is smaller than the estimated upper bound, so it is not strange to see that our branch algorithm is more efficient than the F4 algorithm for such problems.

5 Conclusion

In this paper, we present an implementation of the branch Gröbner bases algorithm and analyze the complexity briefly. The experimental results show that for both randomly generated systems and a class of Stream Ciphers problems, our branch algorithm has better efficiency than the F4 algorithm in Magma. However, there are some problems that are still not solved completely, for example, how to find a general strategy for all problems and how to anticipate the number of branches before the computation, and these can be investigated in the future.

Acknowledgments We thank Professor Xiaoshan Gao for his useful suggestions and Zhenyu Huang for discussing the programming codes.

References

1. Wu, W.T.: Basic principles of mechanical theorem-proving in elementary geometries. J. Sys. Sci. Math. Sci. **4**, 207–235 (1984)
2. Wu, W.T.: Basic principles of mechanical theorem-proving in elementary geometries. J. Autom. Reason. **2**, 221–252 (1986)

3. Faugère, J.C.: A new efficient algorithm for computing Gröbner bases (f4). J. Pure Appl. Algebr. **139**(1), 61–88 (1999)
4. Faugère, J.C.: A new efficient algorithm for computing Grönber bases without reduction to zero (F5). Symbolic and Algebraic Computation, Porc. Conferenz ISSAC 2002, 75–83 (2002)
5. Brickenstein, M., Dreyer, A.: PolyBoRi: A framework for Gröbner basis computations with boolean polynomials. MEGA 2007, Austria (2007)
6. Gao, X.S., Chai, F.J., Yuan, C.M.: A characteristic set method for equation solving in F2 and applications in cryptanalysis of stream ciphers. J. Syst. Sci. Complex. **21**, 191–208 (2008)
7. Sun, Y., Wang, D.K.: Branch Gröbner bases algorithm over boolean ring (Chinese). Preprint (2009)
8. Bardet, M., Faugère, J.C., Salvy, B.: Complexity of Grönber basis computation for Semi-regular overdetermined sequences over F_2 with solutions in F_2. In: Proceedings of the ICPPSS International Conference on Polynomial System Solving Paris, November 24–25-26 2004 in honor of Daniel Lazard (2004)
9. Faugère, J.C., Ars, G.: An Algebraic Cryptanalysis of Nonlinear Filter Generators Using Gröbner Bases. Reserch report 4739, Institut National de Recherche en Informatique et en Automatique, Lorraine (2003)