Deterministic Algorithms for Solving Boolean Polynomial Equations Based on Channel Coding Theory

WU, Guangfu; LV, Yijie; HE, Daojing; HE, Jiguang; ZHOU, Huan; CHAN, Sammy

Published in:
IEEE Access

Published: 01/01/2020

Document Version:
Final Published version, also known as Publisher’s PDF, Publisher’s Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/ACCESS.2020.2971393

Publication details:

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.
Deterministic Algorithms for Solving Boolean Polynomial Equations Based on Channel Coding Theory

GUANGFU WU, YIJIE LV, DAOJING HE, JIGUANG HE, (Member, IEEE), HUAN ZHOU, AND SAMMY CHAN, (Member, IEEE)

1Department of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 330006, China
2Department of Software Engineering, East China Normal University, Shanghai 200241, China
3Centre for Wireless Communications, University of Oulu, 90014 Oulu, Finland
4Department of Electronic Engineering, City University of Hong Kong, Hong Kong 99077

ABSTRACT Solving the satisfiability problems of Boolean polynomial equations is still an open challenge in the fields of mathematics and computer science. In this paper, our goal is to propose a non-algebraic method for solving maximal Boolean polynomial equations (Max-PoSSo problem). By leveraging channel coding theory and dynamic programming, we propose three deterministic and robust algorithms for solving the satisfiability problems of Boolean polynomial equations. Comparisons are made among the three proposed algorithms and Genetic and Gröbner algorithms. Simulation results show that the proposed algorithms exhibit better performance in terms of the largest number of Boolean polynomials equal to 0 compared to the benchmark schemes in the literature.

INDEX TERMS Boolean polynomial equations, non-algebraic, channel coding theory, deterministic, robust.

I. INTRODUCTION

Solving Boolean polynomial equations is one of the major elements for algebraic side-channel attacks, also known as side channel cryptanalysis, which was proposed by Kocher [1] in the late 1990s. It is an attack method based on information gained from the implementation of a computer system. The algebraic attack is to set the cryptographic information as a variable, then establish a set of polynomial equations by considering the relationship between the known information and the cryptographic information, and finally recover the cryptographic information through solving the set of polynomial equations. Algebraic side-channel attacks combine algebraic attacks with side-channel attacks [2]. Algebraic side-channel attacks reduce the attack complexity via introducing side channel information, thereby improve the attack efficiency. The challenge in algebraic side-channel attacks is that polynomial equations are difficult to solve.

Boolean polynomials were proposed by George Boer in his book The Mathematical Analysis of Logic, and further detailed in An Investigation of the Laws of Thought [3] in the nineteenth century. Solving (general) Boolean polynomial equations [4] is still one of the most difficult problems in the fields of mathematics and computer science [5], [6]. Namely, a set of Boolean polynomials is selected to find variable assignments so that the values of these polynomials are all 0. The satisfiability problem of solving maximal Boolean polynomial equations (Max-PoSSo problem) is an extended problem of solving general Boolean polynomial equations [7]. It falls into the category of NP-hard problems [8] and is widely used in cryptographic probabilistic algebraic attacks and side-channel algebraic attacks. In the process of establishing a polynomial equation set, noise
makes certain mistakes in some terms of the set of polynomial equations (generally constant terms), resulting in partial non-zero polynomials [9]. In order to restore the secret information, the attacker needs to find the largest number of Boolean polynomials equal to 0.

There are two main approaches for dealing with the Max-PoSSo problem in the literature. One is to traverse the variable and reduce the search branch as to find the optimal variable assignment. It mainly uses some algebraic transformations to transform the Max-PoSSo problem into new problems with already-existing solutions [10]. Therefore, how to transform the Max-PoSSo problem into other equivalent problems to make the solution readily obtained is the key of the method. The other approach is fixing the value of each polynomial to solve the corresponding general Boolean polynomial equations. If one can find the polynomial that corresponds to the solution of the general polynomial equations and has the largest number of 0, the solution of the general polynomial equations is taken as the solution of the Max-PoSSo problem [11]. To solve the general polynomial equations, effective algorithms include the method based on Gröbner basis [12], [13], the SAT method [8], random Muti-Bit flipping method [14], and the feature column method [15]. However, how to effectively combine the algebraic solution algorithm with the search strategy to reduce the search branch and improve the efficiency of the single branch needs to be further studied.

In this paper, we propose new deterministic algorithms to solve the Max-PoSSo problem. The algorithms are different from the previous ones in the literature, which are non-algebraic. By leveraging channel coding theory and dynamic programming, three deterministic and robust algorithms are proposed to solve the satisfiability problems of Boolean polynomial equations. By exploiting channel coding theory, the proposed algorithms are capable of making the initial solutions uniformly distributed over the entire space [16]. Then, a better solution can be found by searching within a fixed radius.

II. PROBLEM DESCRIPTION

The satisfiability problem of maximal Boolean polynomial equations is an extended problem of general Boolean polynomial equation. The satisfiability problem of maximal Boolean polynomial equations is expressed in the sequel. Given the following $m$ Boolean polynomial equations with $n$ variables:

\[
\begin{align*}
  f_1(x_0, x_1, \cdots, x_{n-1}) , \\
  f_2(x_0, x_1, \cdots, x_{n-1}) , \\
  \cdots , \\
  f_m(x_0, x_1, \cdots, x_{n-1}) ,
\end{align*}
\]

the Max-PoSSo problem is to find a set of $x_0, x_1, \cdots, x_{n-1}$ in GF(2) to make Boolean polynomial equations $f_1, f_2, \cdots, f_m$ have the largest number of value 0 [17], where GF(2) is the Galois field of two elements, 0 or 1.

III. THE ALGORITHM BASED ON CHANNEL CODING THEORY IS DETERMINISTIC AND ROBUST

Forward error correction (FEC) coding is an important technique in communication. In 1948, Shannon demonstrated in the seminal paper that [18], errors induced by a noisy channel or the storage medium can be reduced to be arbitrarily small without sacrificing the information transmission rate or storage rate [19].

The fundamental principle of error-correcting codes [20] is that the sender encodes a message by adding redundant information. The redundancy allows the receiver to detect or correct a limited number of errors that may occur anywhere in the message, thus messages with errors can be recovered without requiring any retransmission. The message that is allowed to be sent is called a codeword, and the set of codewords forms an error correction code. If the received signal is not a legitimate codeword, it is certain that error(s) occurs in the transmission. It is still possible to recover the information from the error-prone message if the number of errors are within certain limit, which is determined by the minimum Hamming distance of the code [21].

**Definition 1**: A $(n, k)$ linear block code has $2^k$ codewords forming a $k$-dimensional subspace of the vector space of all the $n$-tuples in GF(2).

The modulo-2 addition of any two codewords will be another codeword. Three algorithms based on error correction coding will be presented in the sequel for solving the Max-PoSSo problem. These algorithms are deterministic and robust. The core idea of these three algorithms is that the construction of the linear block code is progressively enhanced in terms of the largest number of Boolean polynomials equal to 0.

Algorithm 1 is described as follows:

1) Construct a $k \times n$ generator matrix (i.e., $G$) of a linear block code.
2) Obtain all the codewords $C = \{c_1, \cdots, c_{2^k}\}$ according to the generator matrix, $|C| = 2^k$. In order to solve Boolean polynomial equations, a generator matrix needs to be constructed in a pre-specified manner.
3) Take one codeword obtained in Step 2 as an initial solution and compute the number of Boolean polynomials equal to 0.
4) Start from the initial solution and set the number of erroneous bits as $r$ (i.e., search radius). Candidate solutions obtained from the initial solution has a Hamming distance less than or equal to $r$. Go through all the candidate solutions $D = \{d_1, d_2, \cdots\}$ with $|D| = \binom{n}{r} + \cdots + \binom{r}{r}$ and compute their corresponding numbers of Boolean polynomials equal to 0. Find the candidate solution with the largest number of Boolean polynomials equal to 0 and set it as a suboptimal solution associated with the codeword in Step 3. In one round of search, until going through all variables within radius $r$ the algorithm will stop.
5) Obtain the largest number of Boolean polynomial equations equal to 0.
Algorithm 1 Based on Coding Theory

Require:
\(k\): Number of rows of the generated matrix;
\(G\): Generator matrix;
\(r\): Search radius;
\(f_1, f_2, \ldots, f_m\): Boolean polynomial equations;
\(\text{gen}(c)\): Generate all the candidates which have Hamming distance less than or equal to \(r\) with \(c\);
\(\text{count}()\): Count the number of \(f_1, f_2, \ldots, f_m\) being 0;

Ensure:
\(\text{Max}: \text{Maximum number of } f_1, f_2, \ldots, f_m \text{ being 0;}

1: for \(j = 1 : 2^k \) do
2: \[D_j = \text{gen}(c);\]
3: for \(m = 1 : \binom{r}{1} + \cdots + \binom{r}{k}\) do
4: \[C(j, 1) = \text{count}(c);\]
5: \(C(j, m + 1) = \text{count}(d_{j,m});\)
6: end for
7: end for
8: \(m = \max(C);\)
9: \(\text{Max} = \max(m);\)
10: return \(\text{Max}\)

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The concept of FEC coding is used to solve the Max-PoSSo problem in this experiment.

As an introductory example, the number of Boolean polynomial equations is set to be 8 and the number of variables is set to be 4. The Boolean polynomial equations are provided in (2).

\[
\begin{align*}
1 & = (x_0 + x_2 + x_3) \ast x_3 + (x_1 + x_2 + 1) \\
2 & = (x_1 + x_2 + x_3) \ast x_3 + (x_2 + x_3) \\
3 & = (x_1 + x_2 + x_0) \ast x_3 + (x_2 + 1) \ast x_2 + x_1 \\
4 & = (x_1 + 1) \ast x_3 + (x_1 + x_3 + x_0) \ast x_3 + 1 \\
5 & = (x_1 + x_2 + x_3 + 1) \ast x_1 + (x_3 + x_2) \\
6 & = (x_0 + x_3) \ast x_3 + (x_0 + x_1 + x_3) \ast x_1 + 1 \\
7 & = (x_3 + x_2) \ast x_0 + (x_2 + x_3 + x_1) \ast x_1 + 1 \\
8 & = (x_3 + x_2 + x_1 + 1) \ast x_3 + (x_1 + x_2) \\
& \quad \ast x_0 + (x_0 + x_2 + x_3) + x_2 \ast x_3 + 1
\end{align*}
\]

Each row of matrix \(A\) is a binary representation of natural numbers from 0 to 7 while each row of matrix \(B\) is a binary representation of odd numbers from 0 to 7. Multiplying \(A\) with \(B\) offers an initial solution (in total 8 solutions). Then, based on each initial solution, we increase the search radius \(r\) until finding the candidate solution with the largest number of Boolean polynomials equal to 0. The experimental results are shown in Table 1, Table 2 and Table 3. Most search results with a radius of 1 in Table 1 cannot satisfy the condition that six polynomials in (2) are equal 0. When the search radius is increased to 2, the number of satisfying polynomials saturates.

\[
A = \begin{pmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
1 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1
\end{pmatrix}
\]

\[
B = \begin{pmatrix}
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1
\end{pmatrix}
\]

Next, we extend Algorithm 1 to more complicated cases, where the number of Boolean polynomials \(m\) is set to 256 and the number of variables \(n\) in all the equations is set to 128. Table 4, and Table 5 show the results of this experiment. As shown in tables, each of the 256 initial solutions satisfies the largest number of Boolean polynomials equal to 0. In the tables, 1:181 means the largest number of Boolean polynomials equal to 0 is 181 for the initial solution.

Table 4 shows the search results of Algorithm 1 with radius 2, and the number of Boolean polynomials that being 0 is between 173 and 200. Table 5 shows the search results of Algorithm 1 with radius 4, and the number of Boolean polynomials being 0 is between 189 and 223. Table 6 presents the statistics of the experimental results. Fig. 1 shows the
TABLE 4. The results of searching radius 2 in Algorithm 1.

<table>
<thead>
<tr>
<th>n</th>
<th>1.190</th>
<th>3.148</th>
<th>7.128</th>
<th>12.148</th>
<th>17.128</th>
<th>22.148</th>
<th>27.128</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>65.184</td>
<td>98.184</td>
<td>131.184</td>
<td>164.184</td>
<td>197.184</td>
<td>230.184</td>
<td>263.184</td>
</tr>
</tbody>
</table>

FIGURE 1. Experimental results with different search radii.

performance of Algorithm 1 (the X-axis represents the initial point of search, and the Y-axis represents the largest number of Boolean polynomials equal to 0).

The number of Boolean polynomials in Table 4 is relatively small because the search radius is only 2. According to Table 6, it can be concluded that the largest value (i.e., the largest number of polynomials equal to 0) and expected value increase with the radius $r$. Meanwhile, the standard deviation is basically stable within a range when the radius changes.

In order to verify the algorithm is deterministic and robust, we further randomly generate initial solutions and provide the new results in Table 7, Table 8, Table 9, Fig. 2 and Fig. 3.

TABLE 5. The results of searching radius 4 in Algorithm 1.

<table>
<thead>
<tr>
<th>n</th>
<th>1.190</th>
<th>3.148</th>
<th>7.128</th>
<th>12.148</th>
<th>17.128</th>
<th>22.148</th>
<th>27.128</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>65.184</td>
<td>98.184</td>
<td>131.184</td>
<td>164.184</td>
<td>197.184</td>
<td>230.184</td>
<td>263.184</td>
</tr>
</tbody>
</table>

320 initial are randomly generated and the largest numbers of Boolean polynomials are calculated accordingly. Comparing to the results in Table 6, the randomly constructed initial solution results in almost the same largest value as long as the search radii are the same. In other words, Algorithm 1 converges regardless of the initial generator matrix in the first step. Therefore, Algorithm 1 is deterministic and robust to find the maximum value of Boolean polynomial equations. When $n$ goes large, our proposed scheme exhibits significant computational complexity gain compared to the existing benchmark scheme.

In order to verify the stability of the algorithm 1, we choose 8 out of 256 suboptimal solutions which have the largest number of Boolean polynomials equal to 0. Then, we construct a new generator matrix with the 8 suboptimal solutions as its rows and continue Step 2 until the largest number of Boolean polynomials equal to 0 is not increasing or the maximum number of iterations is 10.

Algorithm 2 is described as follows:

1. Construct a $k \times n$ generator matrix (i.e., $G$) of a linear block code.
2) Obtain all the codewords \( C = \{c_1, \ldots, c_k\} \) according to the generator matrix, \(|C| = 2^k\).

3) Take one codeword obtained in Step 2 as an initial solution and compute the number of Boolean polynomials equal to 0.

4) Start from the initial solution and set the number of erroneous bits as \( r \) (i.e., search radius). Candidate solutions obtained from the initial solution has a Hamming distance less than or equal to \( r \). Go through all the candidate solutions \( D = \{d_1, d_2, \ldots\} \) with \(|D| = \binom{n}{r_1} + \cdots + \binom{n}{r}\).
Algorithm 2 Iterative Algorithm Based on Coding Theory

Require:
- $k$: Number of rows of the generated matrix;
- $G^{(1)}$: Generator matrix;
- $r$: Search radius;
- $f_1, f_2, \ldots, f_m$: Superscript (1) is the iteration index
- $\text{Iter}_\text{max}$: Maximum number of iterations;
- $\Delta_{\text{max}}$: Increase of the largest number of $f_1, f_2, \ldots, f_m$ being 0, which is initialized as 0.
- $\text{gen}():$ Generate all the candidates which have Hamming distance less than or equal to $r$ with $c$;
- $\text{count}():$ Count the number of $f_1, f_2, \ldots, f_m$ being 0;

Ensure:
- Max: Maximum number of $f_1, f_2, \ldots, f_m$ being 0;

1: while $i < \text{Iter}_\text{max}$ and $\Delta_{\text{max}} > 1$ do
  2:     for $j = 1 : 2^k$ do
  3:         $T_j^{(i)} = \text{gen}(c_j^{(i)})$; $\triangleright$ Superscript (i) is the iteration index
  4:         for $m = 1 : (\binom{j}{1}) + \ldots + (\binom{j}{r})$ do
  5:             $C(j, 1) = \text{count}(c(j)^{(i)})$; $\triangleright C$ is a matrix, collecting the number of polynomials being 0
  6:             $C(j, m + 1) = \text{count}(d(j)^{(i)}_{j,m})$; $\triangleright d(j)^{(i)}_{j,m}$ denotes the $m$-th element in the set $T_j^{(i)}$ at $i$-th iteration
  7:         end for
  8:     end for
  9:     $m_1^{(i)} = \max(C)$; $\triangleright m_1^{(i)}$ is a column vector containing the maximum value of each row
10:    $\max^{(i)} = \max(m_1^{(i)})$; $\triangleright \max^{(i)}$ is the maximum value in $m_1^{(i)}$
11:    Choose $k$ $n$-tuples corresponding to the $k$ largest values in $m_1^{(i)}$ as rows of new generator matrix $G^{(i+1)}$;
12:    $\Delta_{\text{max}} = \max^{(i)} - \text{Max};$
13:    $\text{Max} = \max^{(i)}$;
14: end while
15: return $\text{Max}$

and compute their corresponding numbers of Boolean polynomials equal to 0. Find the candidate solution with the largest number of Boolean polynomials equal to 0 and set it as a suboptimal solution associated with the codeword in Step 3.

5) Repeat Steps 3 and 4 until all the codewords are covered. Choose $k$ out of $2^k$ suboptimal solutions which have the largest number of Boolean polynomials equal to 0. Construct a new generator matrix with the $k$ suboptimal solutions as its rows and continue Step 2 until the largest number of Boolean polynomials equal to 0 is not increasing or the maximum number of iterations is reached.

6) Obtain the largest number of Boolean polynomials equal to 0 and set its corresponding $n$-tuples as the final solution.

The experimental results are shown in the Tables 10, 11, 12. The results of the first, fifth and tenth iterations show that the number of polynomials equal to 0 is stable.

V. COMPARISON WITH OTHER ALGORITHMS

In [22], solving the maximum Boolean polynomial equations is consistent with the problem studied in this paper.

Genetic algorithm is a computational method that simulates natural selection and biological evolution. The optimal solution is searched by simulating the natural evolutionary process. The initial individual is selected based on a fitness function while the optimal individual is obtained in the last generation after crossover and mutation operations [22]. Because of the "premature phenomenon" of the Genetic algorithm, the local convergence rate is slow in the absence of effective heuristic information, and the final solution is...
TABLE 12. The results of searching radius 3 in Algorithm 2 and the number of iterations is 10.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Genetic</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Largest value</td>
<td>182</td>
<td>223</td>
</tr>
</tbody>
</table>

VI. EXTENSION ALGORITHM

In step 4) of Algorithm 2, some candidate solutions may be repetitively chosen. Therefore, a modified algorithm, denoted as Algorithm 3, will be introduced to address the aforementioned drawback.

We explain the potential for further improving the efficiency of solving the Max-PoSSo problem in the sequel. Let the Hamming distance between \( c_1 \) and \( c_2 \) be denoted by \( d \). The location set of different bits between \( c_1 \) and \( c_2 \) is denoted by \( M \), then \( |M| = d \). The location set of the same bits between \( c_1 \) and \( c_2 \) is denoted by \( N \), \( |N| = n - d \). Fig. 4 shows the relationship between \( c_1 \) and \( c_2 \) with complexity \( O(n^2) \), while Gröbner basis algorithm has complexity \( O(2^n) \). It is clear that the complexity of the Gröbner basis algorithm is much higher than that of our proposed algorithm. The Gröbner algorithm can solve general Boolean polynomial equations. However, due to the high complexity of the algorithm, it is necessary to consider the computation efficiency when solving a large number of variable equations. Therefore, Gröbner algorithm is not suitable for solving maximal Boolean polynomial equations.
Algorithm 3 Enhanced Version of Algorithm 2

Require:
- k: Number of rows of the generated matrix;
- G(1): Generator matrix;
- r: Search radius;
- f_1, f_2, ..., f_m: Superscript (1) is the iteration index
- \text{Iter}_{\text{max}}: Maximum number of iterations;
- \Delta_{\text{max}}: Increase of the largest number of f_1, f_2, ..., f_m being 0, which is initialized as 0.
- \text{gen}(c_j|c_{j-1}): Generate all the candidates which have Hamming distance less than or equal to r with c_j conditioned on c_{j-1};
- \text{count}(): Count the number of f_1, f_2, ..., f_m being 0;

Ensure:
- Max: Maximum number of f_1, f_2, ..., f_m being 0;

1: \textbf{while } i < \text{Iter}_{\text{max}} \text{ and } \Delta_{\text{max}} > 1 \textbf{ do}
2: \quad \textbf{for } j = 1 : 2^k \textbf{ do}
3: \quad \quad D_{j}^{(i)} = \text{gen}(c_{j}^{(i)}|c_{j-1}^{(i)});
4: \quad \textbf{for } m = 1 : |D_{j}^{(i)}| \textbf{ do}
5: \quad \quad C_{j}(1) = \text{count}(c_{j}^{(i)});
6: \quad \quad C_{j}(m + 1) = \text{count}(c_{j,m}^{(i)});
7: \quad \textbf{end for}
8: \textbf{end for}
9: \quad m^{(i)} = \max(C);
10: \quad \max^{(i)} = \max(m^{(i)});
11: \quad \text{Choose the } k \text{ } n\text{-tuples corresponding to the } k \text{ largest values in } m^{(i)} \text{ as rows of new generator matrix } G^{(i+1)};
12: \quad i++;
13: \quad \Delta_{\text{max}} = \max^{(i)} - \text{Max};
14: \quad \text{Max} = \max^{(i)};
15: \textbf{end while}
16: \textbf{return } \text{Max}

In this case, the candidate solutions for c_1 and c_2 are overlapping with each other (overlapped area is filled with lines). In Algorithm 2, the candidate solutions for c_1 will be in the ball centered at c_1 with radius r. Similarly, the candidate solutions for c_2 will be in the ball centered at c_2 with radius r. Overlapped area is considered twice. Therefore, in Algorithm 3, only the candidate solutions in the gray area are considered for c_2. The first three steps of Algorithm 3 are the same as Algorithm 2.

The remaining steps of Algorithm 3 are described as follows:

4) Start from the initial solution denoted c_1, and set the number of erroneous bits as r (i.e., search radius). Candidate solutions obtained from the initial solution have a Hamming distance less than or equal to r. Compute the number of Boolean polynomials equal to 0 for all the candidate solutions. The candidate solution with largest number of Boolean equation equal to 0 is taken as a suboptimal solution associated with the corresponding codeword c_1.

5) In order to avoid repeated search for c_2, choose u bits in the location set M and s bits in the location set N. Note that u and s must satisfy the condition r + u − d < s ≤ r − u and 0 < u ≤ d/2. The candidate solution with largest number of Boolean equation equal to 0 is taken as a suboptimal solution associated with the corresponding codeword c_2.

6) Repeat steps 3), 4) and 5) until all the codewords are covered. Choose k out of 2^k suboptimal solutions which have the largest number of Boolean polynomials equal to 0. Construct a new generator matrix with the k suboptimal solutions as its rows and continue step 2) until the largest number of Boolean polynomials equal to 0 is not increasing or the largest number of iterations is reached.

7) Obtain the largest number of Boolean polynomials equal to 0 and set its corresponding n-tuples as the final solution.

The experiment in Section IV is repeated by using Algorithm 3. In order to evaluate the performance improvement of the algorithm, we consider the actual running time as a performance metric for comparison. It turns out that Algorithm 3 has the same performance as Algorithm 1 and Algorithm 2, while the running time of Algorithm 3 is 20% less than that of Algorithm 1 and Algorithm 2.

VII. CONCLUSION

In this paper, a non-algebraic method has been proposed to solve the maximal Boolean polynomial. Three deterministic and robust algorithms have been proposed to solve the problem of the satisfiability of Boolean polynomial equations based on coding theory. As the search radius r increases, the number of the Boolean polynomials equal to 0 increases. Algorithm 3 has the same performance as Algorithm 1 and Algorithm 2, while Algorithm 3 runs about 20% faster than Algorithm 1 and Algorithm 2. The number of Boolean polynomials equal to 0 obtained by the proposed algorithms has been significantly increased compared to the benchmark scheme in [21]. Compared to the randomness of genetic algorithm, our proposed algorithms are more deterministic and robust. The complexity of the proposed algorithms is much less than Gröbner algorithm. In addition, the proposed algorithms are capable of solving non-deterministic hard problem in a more effective way.

REFERENCES


