Algorithmic techniques in algebraic cryptanalysis and their applications

Makarim, R.H.

7 Solving Binary Puzzles using Gröbner Bases Algorithms

Contents

This chapter presents an application of Gröbner bases to solve binary puzzles.* These are Sudoku-like puzzles where the entries are taken from the set $\{0, 1\}$. In this chapter we study the reduction of the problem of solving a binary puzzle into a problem of solving a system of multivariate polynomial equations over three different rings. The first approach reduces the problem into the problem of solving system of polynomials over $\mathbb{F}_2$. This is a natural approach since we can view each entry of a binary puzzle as an element of $\mathbb{F}_2$. The second approach reduces the problem into a system of polynomials over $\mathbb{Q}$ or $\mathbb{Z}$. The later approach decreases the degree of some polynomials into linear ones at the expense of defining the polynomials over different rings. The solutions for these system of equations correspond to the correct configurations for the binary puzzle (provided that a solution exists). Thus, we can run Gröbner bases algorithms on these systems of equations to obtain a solution for the binary puzzle. We compare the performance of Gröbner bases algorithm in Magma and SageMath to solve systems of equations from binary puzzle of various sizes over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$.† The result of this chapter is an extension of the published work in collaboration with Utomo [UM17].

We first give an overview of binary puzzles in Section 7.1. This is followed by short description of Constraint Satisfaction Problem (CSP) in Section 7.2. In Section 7.3 we give an overview of prior related works. We then describe necessary notations and preliminaries in Section 7.4. Most of the contents of Section 7.4 is related to the theory of cryptographic Boolean functions, which can be found in [Car10]. The reduction of solving a binary puzzle into solving a system of polynomials over $\mathbb{F}_2$ is described in Section 7.5. This is followed by another reduction into a system polynomials over $\mathbb{Q}$ and $\mathbb{Z}$ in Section 7.6. Finally we conclude this chapter by comparing the performance and memory usage of existing Gröbner bases implementations on various sizes of binary puzzles in Section 7.7.

* also known as Takuzu/Binero/Bineiro/Zernero/Binary Sudoku.

† We could not use M4GB in this case since it lacks optimization for polynomials over $\mathbb{F}_2$ and it does not support Gröbner bases computation over $\mathbb{Z}$ and $\mathbb{Q}$.

7.1 Overview of Binary Puzzles

Binary puzzles are Sudoku-like puzzles in which the value in each cell is taken from the set $\{0, 1\}$ instead of $\{0, 1, \ldots, 9\}$. A binary puzzle is represented by an $n \times n$ square matrix where $n \geq 4$. For some entries explicit values are initially given, while the remaining entries are left undetermined. The goal of a player is to fill every entry in the puzzle while observing the following constraints:

1. (Three-value constraint) There exist no three consecutive entries, both vertically and horizontally, which are filled with the same value,
2. (Balancedness constraint) every row and column is *balanced*, i.e., in each row and each column the number of ones and zeros must be equal,
3. (Distinctness constraint) there exist no two equal rows. Similarly, there exist no two columns that are equal.

An example of initial configuration of a binary puzzle and its corresponding solution is available in Figure 13. The gray-colored entries indicate that their values are undetermined. The black and white cells represent zeroes and ones respectively (or vice versa).

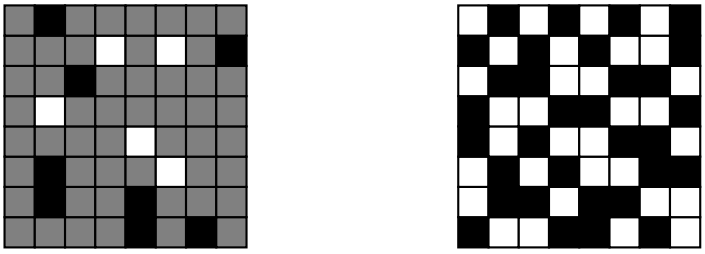

Figure 13: *An example of initial configuration of an 8×8 binary puzzle (left) and its corresponding solution (right). Note that multiple solutions may exist depending on the size and the initial configuration.*

7.2 Constraint Satisfaction Problem

The problem of solving a binary puzzle can be seen as a *constraint satisfaction problem* (CSP) [RN10, Chapter 6]. A CSP consists of the following three components

1. A set of variables $X = \{X_1, \ldots, X_n\}$,
2. A set of domains $D = \{D_1, \ldots, D_n\}$ where each domain D_i is the set of permitted values for the corresponding variable X_i,
3. A set of constraints $C = \{C_1, \ldots, C_k\}$ where each constraint C_i consists of a pair $(S, \texttt{rel})$ where S is an element of the power set of X which tells the variables that participate in the constraint and `rel` is a relation that defines the values that the variables in S can take on.

We speak about the relation `rel` in a loose fashion since there are multiple ways to describe it. A relation can be described as an explicit list of all tuples of

values that satisfy the constraint. For instance, a constraint that is satisfiable when two variables X_1, X_2 are not equal and both have the domain $\{0,1\}$ can be described as $(\{X_1, X_2\}, \{(0,1),(1,0)\})$. Another approach to describe the relation `rel` is by using a Boolean formula. For instance, the constraint in the previous example can be described as $(\{X_1, X_2\}, X_1 \neq X_2)$.

Described as a CSP, the problem of solving an $n \times n$ binary puzzle consists of n^2 variables. For $1 \leq i, j \leq n$, we denote by $x_{i,j}$ the variable that represents the entry at row i and column j. The domain $D_{i,j}$ of each variable $x_{i,j}$ is equal to $D_{i,j} = \{0,1\}$. For the three-value constraint, the number of CSP constraints for each row or column is equal to $n-2$. In total, there are $2n(n-2)$ constraints to describe it, each involving three variables. The number of constraints needed to describe the balancedness constraint is $2n$ where each of them involves n variables. Lastly, the number of constraints to describe the distinctness constraint is $2 \cdot \binom{n}{2} = n(n-1)$ where each of them involves $2n$ variables. In total, there are $3n(n-1)$ constraints to describe an $n \times n$ binary puzzle as a CSP.

7.3 Related Work

The complexity of solving binary puzzles was studied in [dB12]. Utomo and Pellikaan studied binary puzzle as a constrained array and analyzed its properties from erasure correcting point of view [UP15]. One direction to solve binary puzzles was also proposed by the same author in [UP15, UP17] by expressing the constraints as Boolean satisfiability problem in conjunctive normal form. An off-the-shelf SAT solvers (e.g., CryptoMiniSAT [Soo16]) was employed to find a solution for a binary puzzle. Utomo also proposed another approach that models the problem of solving binary puzzles as an instance of satisfiability modulo theory [Uto17, Mon16]. A physical zero-knowledge proof to verify the solution of a binary puzzle was developed in [BDDL16, Section 3].

The use of a system of multivariate polynomials to represent the constraints in a finite-domain CSP was studied in [JJGvD13]. One primary advantage in using multivariate polynomials to represent constraints in a CSP is that it allows different forms of constraint to be treated in a uniform way. The polynomials in [JJGvD13] are treated strictly as elements of $\mathbb{C}[x_1, \ldots, x_n]$. In order to restrict the domain of each variable to a finite set, say $D \subset \mathbb{C}$, the polynomials of the form $\prod_{j \in D}(x - j)$ are included in the system of equation. The paper also demonstrated how different types of constraints (constant, disjunctive, inequality, etc) can be expressed as multivariate polynomials. Finally, the authors suggested to use Gröbner bases algorithms to solve a system of equations that represents a CSP.

7.4 Notations and Preliminaries

We denote by $\mathrm{wt}(u)$ the *Hamming weight* of $u \in \{0,1\}^n$, that is the number of nonzero components of u. The vector $v = (v_1, \ldots, v_n) \in \mathbb{F}_2^n$ is said to be covered by $w = (w_1, \ldots, w_n) \in \mathbb{F}_2^n$ (or w covers v), denoted by $v \preceq w$, if $v_i \leq w_i$ for all $i \in \{1, \ldots, n\}$. For nonnegative integers $p = \sum_{i=0}^{n} p_i 2^i, q = \sum_{i=0}^{n} q_i 2^i$ we say that $p \preceq q$ if $(p_0, p_1, \ldots, p_n) \preceq (q_0, q_1, \ldots, q_n)$ where $n = \lceil \log_2(\max(p,q)) \rceil$ and $p_i, q_i \in \{0,1\}$.

An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The n-variable *pseudo Boolean function* corresponding to f is the $\mathbb{Z}$-valued function φ_f on $\mathbb{F}_2^n$. By canonical ring embedding from $\mathbb{Z}$ to $\mathbb{Q}$, φ_f can also be seen as $\mathbb{Q}$-valued function. The Boolean function f can be represented as an n-variable polynomial in the ring $\mathbb{F}_2[x_1, \ldots, x_n]$

$$f = \sum_{\substack{u \in \{0,1\}^n \\ u=(u_1,\ldots,u_n)}} a_u x_1^{u_1} \cdots x_n^{u_n} \tag{7.1}$$

where $a_u \in \mathbb{F}_2$. The representation in (7.1) is called the *algebraic normal form* (ANF) of f. The coefficients a_u is computed using the following proposition.

Proposition 7.1. *Let $\sum_{u\in\{0,1\}^n} a_u x_1^{u_1} \cdots x_n^{u_n}$ be the ANF of an n-variable Boolean function f. For all $u \in \{0,1\}^n$, we have*

$$a_u = \sum_{\substack{x \in \mathbb{F}_2^n \\ x \preceq u}} f(x).$$

Proof. See [Car10, Proposition 1]. ■

Similarly, one can obtain the polynomial representation of a Boolean function f in $\mathbb{Z}[x_1, \ldots, x_n]$. We refer to such polynomial as the *numerical normal form* (NNF) of f.

Proposition 7.2. *Let $\sum_{u\in\{0,1\}^n} a_u x_1^{u_1} \cdots x_n^{u_n}$ be the NNF of an n-variable Boolean function f where $a_u \in \mathbb{Z}$. For all $u \in \{0,1\}^n$ we have*

$$a_u = (-1)^{\mathrm{wt}(u)} \sum_{\substack{x \in \mathbb{F}_2^n \\ x \preceq u}} (-1)^{\mathrm{wt}(x)} \varphi_f(x).$$

Proof. See [Car10, Proposition 4]. ■

Remark 7.3. *Note that the ANF of a Boolean function can be computed from its NNF by reducing the coefficients modulo* 2.

Definition 7.4. *An n-variable Boolean function f is called symmetric if its output is invariant under any permutation of its input bits, that is*

$$f(x_1, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

for all permutation $\pi : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$.

From the definition above, the output of a symmetric Boolean function depends only on the Hamming weight of its input. The following notion defines an efficient representation for a symmetric Boolean function.

Definition 7.5. *For a symmetric n-variable Boolean function f, we associate a function $v_f : \{0, \ldots, n\} \mapsto \mathbb{F}_2$ such that $f(x) = v_f(\mathrm{wt}(x))$ for all $x \in \mathbb{F}_2^n$. The sequence $v(f) = (v_f(0), \ldots, v_f(n))$ is called the simplified value vector of f.*

Definition 7.6 (Elementary Symmetric Polynomials)**.** *The i-th elementary symmetric polynomials $\sigma_{i,n}$ in n-variables $x_1, \ldots, x_n$ is defined as*

$$\sigma_{i,n}(x_1, \ldots, x_n) = \sum_{1 \leq j_1 < j_2 < \cdots < j_i \leq n} x_{j_1} \cdots x_{j_i}.$$

Proposition 7.7. *An n-variable Boolean function f is symmetric if and only if its algebraic normal form can be written as*

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{n} \lambda_f(i) \cdot \sigma_{i,n}(x_1, \ldots, x_n), \quad \lambda_f(i) \in \mathbb{F}_2.$$

Proof. See [CV05, Proposition 1]. ■

Definition 7.8. *The coefficients of the ANF of a symmetric Boolean function f can be represented by the sequence of $(n+1)$-bit $\lambda(f) = (\lambda_f(0), \ldots, \lambda_f(n))$ called the simplified ANF vector of f.*

Proposition 7.9. *Let f be an n-variable symmetric Boolean function. The simplified ANF vector $\lambda(f)$ is related with its simplified value vector $v(f)$ by*

$$\lambda_f(i) = \sum_{k \preceq i} v_f(k).$$

Proof. See [CV05, Proposition 2]. ■

7.5 Polynomial Equations over $\mathbb{F}_2$

In this work, we investigate the use of systems of polynomials to specify constraints of binary puzzles. Since the domain of each variable is $\{0, 1\}$, the first natural approach is to treat this set as the binary field $\mathbb{F}_2$. This allows each constraint to be viewed as an k-variable Boolean function, where k is the number of variables involved in the constraint. For instance, the three-value constraint is a 3-variable Boolean function, whereas the balancedness constraint is an n-variable Boolean function. We say that a Boolean function allows a specific value assignment for its variables if and only if the Boolean function output equals 0. This is in line with the notion of a solution to a system of polynomials.

Recall that an n-variable Boolean function can be represented as a multivariate polynomial in (7.1) over $\mathbb{F}_2$. We use this observation and Proposition 7.1 to derive the polynomial that represents the three-value constraint.

Theorem 7.10. *Let x, y, z be variables that represent three adjacent cells (horizontally or vertically) of an $n \times n$ binary puzzle. The three-value constraint of an $n \times n$ binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$xy + xz + x + yz + y + z + 1. \tag{7.2}$$

Proof. Let f_1 be a Boolean function that represent the three-value constraint of binary puzzle. By definition f_1 is defined as

$$f_1(x, y, z) = \begin{cases} 1 & \text{if } (x, y, z) = (0, 0, 0) \textbf{ or } (x, y, z) = (1, 1, 1) \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

By Proposition 7.1, the coefficients a_u of the ANF of f_1 can be computed as:

$$a_u = \begin{cases} 0 & \text{if } u = (1,1,1) \\ 1 & \text{otherwise.} \end{cases}$$

Therefore $f_1(x,y,z) = xy + xz + x + yz + y + z + 1$. ■

We also use a similar approach to derive the polynomial representation for the balancedness constraint. We first recall the following result.

Lemma 7.11 (Lucas' Theorem [Luc78])**.** *Let n, m be nonnegative integers. A binomial coefficient $\binom{m}{n}$ is divisible by a prime p if and only if at least one of the digits in p-ary expansion of n is greater than the corresponding p-ary digit of m.*

Corollary 7.12. *Let n, m be nonnegative integers. A binomial coefficient $\binom{m}{n}$ is not divisible by 2 if and only if $n \preceq m$.*

Theorem 7.13. *Let $x_1, \ldots, x_n$ be variables that represent the n cells in a single row/column of an $n \times n$ binary puzzle. The balancedness constraint of binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$1 + \sum_{\substack{u \in \mathbb{F}_2^n \setminus \{0\} \\ n/2 \preceq \mathrm{wt}(u)}} x_1^{u_1} \cdots x_n^{u_n}. \tag{7.4}$$

Proof. Let $f_2 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a Boolean function that represents the balancedness constraint. The function f_2 is defined as

$$f_2(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \mathrm{wt}((x_1, \ldots, x_n)) = n/2 \\ 1 & \text{otherwise.} \end{cases}$$

By Proposition 7.1, the constant coefficient a_0 in the ANF of f_2 is equal to $a_0 = f_2(0) = 1$. For the coefficients a_u such that $u \in \mathbb{F}_2^n \setminus \{0\}$, we will handle the cases for $\mathrm{wt}(u) < n/2$ and $\mathrm{wt}(u) \geq n/2$ separately.

For all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) < n/2$ we define the set $S_u = \{w \in \mathbb{F}_2^n : w \preceq u\}$. By Proposition 7.1 , we have $a_u = 0$ since $f_2(w) = 1$ for all $w \in S_u$ and the cardinality of S_u is even.

We will now determine the coefficients a_u for all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) \geq n/2$. Let $\mathrm{Supp}(u) = \{i : u_i \neq 0\}$ denotes the support of u. The number of $w \in \mathbb{F}_2^n$ such that $\mathrm{wt}(w) = n/2$ and $w \preceq u$ is equal to $\binom{|\mathrm{Supp}(u)|}{n/2} = \binom{\mathrm{wt}(u)}{n/2}$. Therefore, $a_u = 1$ if and only if $\binom{\mathrm{wt}(u)}{n/2}$ is odd, i.e., if and only if $n/2 \preceq \mathrm{wt}(u)$ by Corollary 7.12. Therefore,

$$f_2(x_1, \ldots, x_n) = 1 + \sum_{\substack{u \in \mathbb{F}_2^n \setminus \{0\} \\ n/2 \preceq \mathrm{wt}(u)}} x_1^{u_1} \cdots x_n^{u_n}.$$

■

Our next aim is to simplify the polynomial representation for the balancedness constraint by taking into account some of the properties of the corresponding Boolean function. Note that the Boolean function that represents the balancedness constraint of binary puzzle is symmetric. Following this observation, we derive the following result.

Theorem 7.14. *Let $x_1, \ldots, x_n$ be variables that represent n entries in a single row/column of an $n \times n$ binary puzzle. The balancedness constraint of binary puzzle is represented by the following polynomial over $\mathbb{F}_2$*

$$1 + \sum_{\substack{i=1 \\ n/2 \preceq i}}^{n} \sigma_{i,n}(x_1, \ldots, x_n).$$

Proof. Let $f_2 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be the Boolean function that represents the balancedness constraint of binary puzzle. Since f_2 is a symmetric function, by Proposition 7.7 the algebraic normal form of f_2 can be written as

$$f_2(x_1, \ldots, x_n) = \sum_{i=0}^{n} \lambda_{f_2}(i) \cdot \sigma_{i,n}(x_1, \ldots, x_n), \quad \lambda_{f_2}(i) \in \mathbb{F}_2.$$

The coefficients $\lambda_{f_2}(i)$ can be computed from the simplified value vector v_{f_2} of f_2. The function v_{f_2} is defined as

$$v_{f_2}(i) = \begin{cases} 0 & \text{if } i = n/2, \\ 1 & \text{otherwise.} \end{cases}$$

Recall that by Proposition 7.9, $\lambda_{f_2}(i) = \sum_{k \preceq i} v_{f_2}(k)$. For a nonzero i, the cardinality of the set $\{k \mid k \preceq i, \forall 0 \leq k \leq n\}$ is even. Because $v_{f_2}(i) = 0$ only for $i = n/2$, then for all nonzero i, we have

$$\lambda_{f_2}(i) = \begin{cases} 0 & n/2 \npreceq i \\ 1 & n/2 \preceq i. \end{cases}$$

For $i = 0$, by definition $\lambda_{f_2}(0) = 1$. Therefore

$$f_2(x_1, \ldots, x_n) = 1 + \sum_{\substack{i=1 \\ n/2 \preceq i}}^{n} \sigma_{i,n}(x_1, \ldots, x_n).$$

■

Finally, the following theorem gives a polynomial representation for the distinctness constraint of binary puzzle. The proof is straightforward from the definition of the distinctness constraint.

Theorem 7.15. *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be variables that represent two distinct rows (or columns) of an $n \times n$ binary puzzle. The polynomial representation over $\mathbb{F}_2$ for the distinctness constraint of the binary puzzle is given by*

$$\prod_{i=1}^{n} (x_i + y_i + 1). \tag{7.5}$$

For $1 \leq i, j \leq n$, let $x_{i,j}$ be variable that represent the value at row i and column j of an $n \times n$ binary puzzle. The system of equations over $\mathbb{F}_2$ that represent an $n \times n$ binary puzzle is

$$
\begin{array}{ll}
x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} + x_{i,j+2} & \\
\qquad +x_{i,j}x_{i,j+1} + x_{i,j+1} + x_{i,j} + 1, & 1 \le i \le n, 1 \le j \le n-2 \\
x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} + x_{i+2,j} & \\
\qquad +x_{i,j}x_{i+1,j} + x_{i+1,j} + x_{i,j} + 1, & 1 \le j \le n, 1 \le i \le n-2 \\
1 + \sum\limits_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{i,1}, \ldots, x_{i,n}), & 1 \le i \le n \\
1 + \sum\limits_{\substack{k=1 \\ n/2 \preceq k}}^{n} \sigma_{k,n}(x_{1,j}, \ldots, x_{n,j}), & 1 \le j \le n \\
\prod\limits_{k=1}^{n}(x_{i,k} + x_{j,k} + 1), & 1 \le i < j \le n \\
\prod\limits_{k=1}^{n}(x_{k,i} + x_{k,j} + 1), & 1 \le i < j \le n \\
x_{i,j}^2 + x_{i,j} & 1 \le i \le n, 1 \le j \le n.
\end{array}
$$

The polynomials of the form $x_i^2 + x_i$ are added to the systems in order to eliminate possible solutions in the set $\overline{\mathbb{F}}_2 \setminus \mathbb{F}_2$, where $\overline{\mathbb{F}}_2$ is the algebraic closure of $\mathbb{F}_2$. If the value of t entries of an $n \times n$ binary puzzle are initially given, the number of variables in the system can be reduced to $n - t$ by substituting the corresponding variables with their given values.

7.6 Polynomial Equations over $\mathbb{Q}$ (or $\mathbb{Z}$)

One of the important parameters that determines the complexity of solving system of equations is the degree of the polynomials in the system. In some cases, such as the system of equations from block ciphers (see Subsection 2.3.1) and in the proof of NP-completeness of MQ over $\mathbb{F}_2$ (Proposition 2.4), the degree of polynomials in the system can be reduced by introducing some auxiliary variables. In the case of binary puzzles, we ask if it is possible to reduce the degree of some equations without introducing additional variables in the system.

Although it is natural to express all three constraints of a binary puzzle as polynomials over $\mathbb{F}_2$, one may observe that the polynomials representing the balancedness constraint can be simplified by working over a different ring, such as $\mathbb{Q}$ or $\mathbb{Z}$.

Theorem 7.16. *Let $x_1, \ldots, x_n$ be variables that represent n cells in a single row/column of an $n \times n$ binary puzzle. The second constraint of binary puzzle is represented by the following polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$)*

$$\sum_{i=1}^{n} x_i - n/2. \tag{7.6}$$

Similarly, we can also express the distinctness constraint as polynomials over $\mathbb{Q}$ (or $\mathbb{Z}$) in the following theorem.

Theorem 7.17. *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be variables that represent two distinct rows (or columns) of an $n \times n$ binary puzzle. The polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents the distinctness constraint of a binary puzzle is given by*

$$\prod_{i=1}^{n}(x_i + y_i - 1). \tag{7.7}$$

The remaining step is to obtain the polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents the three-value constraint. While this may not be obvious from the definition of the constraint, we can derive the polynomial representation over $\mathbb{Q}$ (or $\mathbb{Z}$) from the Boolean function (7.3) that represent the first constraint.

Theorem 7.18. *Let x, y, z be variables that represent three adjacent cells (horizontally or vertically) of an $n \times n$ binary puzzle. The three-value constraint of binary puzzle is represented by the following polynomial over $\mathbb{Q}$ (or $\mathbb{Z}$)*

$$yz + xz - z + xy - y - x + 1 \tag{7.8}$$

Proof. Let φ_{f_1} be the pseudo Boolean function of f_1 in (7.3). The numerical normal form of f_1 over $\mathbb{Q}$ (or $\mathbb{Z}$) is the polynomial that represent the first constraint of binary puzzle. The coefficients of the NNF of f_1 can be computed using φ_{f_1} and Proposition 7.2. One can verify that the NNF of f_1 is equal to (7.8). ■

In order to eliminate superfluous solutions, i.e., solutions that do not belong to the set $\{0,1\} \subset \mathbb{Q}$ (or $\{0,1\} \subset \mathbb{Z}$), we also add polynomials of the form $x_{i,j}^2 - x_{i,j}$ to the system. Thus, the system of equations over $\mathbb{Q}$ (or $\mathbb{Z}$) that represents an $n \times n$ binary puzzle is equal to

$$\begin{aligned}
&x_{i,j+1}x_{i,j+2} + x_{i,j}x_{i,j+2} - x_{i,j+2} \\
&\qquad + x_{i,j}x_{i,j+1} - x_{i,j+1} - x_{i,j} + 1, && 1 \le i \le n, 1 \le j \le n-2 \\
&x_{i+1,j}x_{i+2,j} + x_{i,j}x_{i+2,j} - x_{i+2,j} \\
&\qquad + x_{i,j}x_{i+1,j} - x_{i+1,j} - x_{i,j} + 1, && 1 \le j \le n, 1 \le i \le n-2 \\
&\sum_{j=1}^{n} x_{i,j} - n/2, && 1 \le i \le n \\
&\sum_{i=1}^{n} x_{i,j} - n/2, && 1 \le j \le n \\
&\prod_{k=1}^{n} (x_{i,k} + x_{j,k} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&\prod_{k=1}^{n} (x_{k,i} + x_{k,j} - 1), && 1 \le i \le n-1, i+1 \le j \le n \\
&x_{i,j}^2 - x_{i,j} && 1 \le i \le n, 1 \le j \le n.
\end{aligned}$$

The system of equations over $\mathbb{Q}$ (or $\mathbb{Z}$) of an $n \times n$ binary puzzle serves as an example where the degree of some polynomials in the system can be reduced without introducing additional variables. However, this comes at the cost of defining the polynomials over a different ring.

7.7 Solving Binary Puzzles using Gröbner Bases

In order to understand the applicability of Gröbner bases algorithm to solve binary puzzles, it is necessary to perform some practical experiments. The experiments must cover the system of equations of binary puzzle over $\mathbb{F}_2$, $\mathbb{Q}$, and $\mathbb{Z}$. We will compare the performance and memory usage of existing implementation of Gröbner bases algorithm in various computer algebra system. These results can shed some lights in understanding the cost of reducing the degree of the polynomials when defined over $\mathbb{F}_2$, at the expense of defining them over $\mathbb{Q}$ or $\mathbb{Z}$.

We ran the experiments on binary puzzles of size $n = 6, 8, 10, 12, 14$. For each n, we took the first five (5) samples of binary puzzles classified under "very hard" category from `http://www.binarypuzzle.com/`. The experiments were executed on a single core Intel(R) Xeon(R) CPU E5-4640 0 @ 2.40GHz equipped with 128GB of RAM.

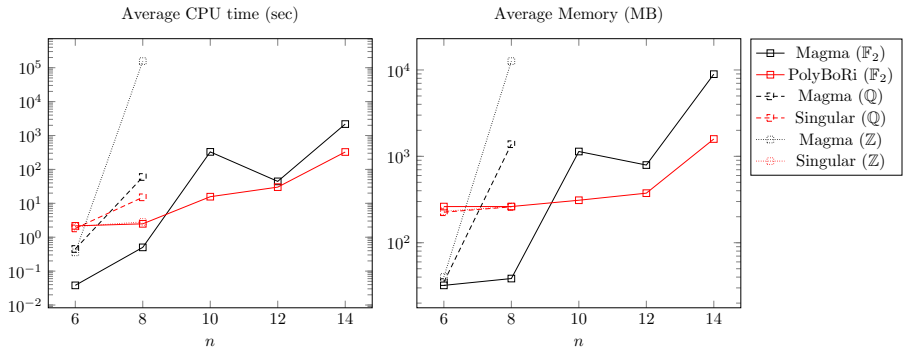

Figure 14: *Average CPU time and memory usage of various implementations of Gröbner bases algorithm to solve polynomial systems from $n \times n$ binary puzzles.*

For system of polynomials over $\mathbb{F}_2$, we compared the implementation of Gröbner bases algorithm in Magma [BCP97] and PolyBoRi/BRiAl* [BD09a]. Magma provided Gröbner bases algorithm for Boolean polynomials since version 2.15.† PolyBoRi/BRiAl is an open-source implementation of Gröbner basis algorithm with specialized data-structure for Boolean polynomials using Zero-Suppressed Binary Decision Diagram (ZDD) [Min95]. We use the implementation of PolyBoRi/BRiAl inside SageMath [The18]. The results of the experiment are available in Table 17.

For system of equations that represent 6×6 and 8×8 puzzles over $\mathbb{F}_2$, the implementation of Gröbner bases algorithm in Magma consistently solved those systems faster with less memory usage compared to PolyBoRi/BRiAl. However, PolyBoRi/BRiAl solved the systems from binary puzzles of size $n \geq 10$ much faster and with much less memory usage than Magma, often by significant factor. For instance, the system of equations from 10×10 binary puzzles were solved 1.14 up to 43.94 times faster and it used 1.52 up to 7.66 less memory than Magma (with exception on the first and the last puzzle where the memory usage of Magma is less than PolyBoRi/BRiAl). This efficiency of PolyBoRi/BRiAl is even more obvious for system of equations from 14×14 binary puzzles, where it solved the second and the third cases 418 and 226 times faster than Magma, respectively. For the same case, the efficiency of its memory usage reached a factor of 52 and 37 times less than Magma. For $n = 12$, Magma performed faster than PolyBoRi/BRiAl only for the fourth case by a factor of 1.86 but with 1.06 times more memory. The remaining four cases were solved by PolyBoRi/BRiAl by a factor ranging from 3.49 up to 10.16 times faster and up to 3.3 times less memory usage than Magma.

However, it is not possible to draw a conclusion why PolyBoRi/BRiAl performed significantly better than Magma to solve system of equations from binary

* `https://github.com/BRiAl/BRiAl`
† `http://magma.maths.usyd.edu.au/magma/handbook/text/1213#13600`

puzzles with $n \geq 10$ over $\mathbb{F}_2$. While much can be said about the internal implementation of PolyBoRi/BRiAl, this is not the case with Magma due to its closed-source nature.

For system of equations over $\mathbb{Q}$ and $\mathbb{Z}$, we compared the implementations of Gröbner bases algorithm in Magma and Singular [DGPS18]. The experiments were limited to $n = 6, 8$ and the results are given in Table 18 and Table 19. For $n \geq 10$, the computation of Magma exceeded two-weeks time for one single instance of polynomial system and some of the computation of Singular were terminated because the process ran out of memory. At this point we can already see that reducing the degree of polynomials over $\mathbb{F}_2$ at the expense of defining them as polynomials over $\mathbb{Q}$ and $\mathbb{Z}$ does not gain any practical advantage to compute the Gröbner bases of the corresponding polynomial ideal. We summarized the result in Figure 14.

Note that solving a binary puzzle using Gröbner bases algorithm is not the most efficient method to obtain a solution for a binary puzzle. In [UM17], the authors demonstrated that backtrack-based search and SAT solvers are better approach to solve a binary puzzle than using Gröbner bases algorithm. The advantage of using Gröbner bases is it allows a parametrization of the set of solutions and it helps to show which cells from a configuration of a binary puzzle that has multiple solutions.

7.8 Scope and Limitations

The scope of the modelling of binary puzzles presented in this chapter, formulated as systems of equations, does not allow a direct comparison with M4GB algorithm. This is because the current implementation of the M4GB algorithm is tailored for quadratic, dense systems of equations defined over an odd prime field or a proper extension of $\mathbb{F}_2$. In contrast, the systems of equations arising from binary puzzles are sparse, of higher degree, and defined over the binary field, the integer ring, or the field of rational number.

Furthermore, a complexity analysis of solving binary puzzles via Gröbner bases is intentionally omitted. Such analysis would only be meaningful for specific instances of the puzzle, and since different instantiations yield different systems of equations, the resulting complexity estimates would vary accordingly. Consequently, this approach would not provide a representative measure of the average computational complexity of solving binary puzles using Gröbner bases algorithm.

6×6				8×8			
PolyBoRi		Magma		PolyBoRi		Magma	
Time	Memory	Time	Memory	Time	Memory	Time	Memory
2.19 s	262.6 MB	0.04 s	32.1 MB	2.45 s	263.3 MB	1.73 s	64.1 MB
2.12 s	262.4 MB	0.06 s	32.1 MB	2.35 s	263 MB	0.18 s	32.1 MB
2.22 s	261.4 MB	0.04 s	32.1 MB	2.49 s	262.3 MB	0.14 s	32.1 MB
2.10 s	261.3 MB	0.03 s	32.1 MB	2.48 s	261.8 MB	0.32 s	32.1 MB
2.19 s	263 MB	0.02 s	32.1 MB	2.65 s	261.9 MB	0.14 s	32.1 MB

10×10			
PolyBoRi		Magma	
Time	Memory	Time	Memory
3.84 s	264.4 MB	15.45 s	250.8 MB
61.7 s	490.9 MB	1325.1 s	3761.1 MB
4.23 s	264.7 MB	51.38 s	401.5 MB
5.68 s	267.6 MB	249.56 s	1074.8 MB
2.94 s	263.9 MB	3.36 s	198.9 MB

12×12			
PolyBoRi		Magma	
Time	Memory	Time	Memory
17.34 s	320.9 MB	60.46 s	1011.6 MB
4.10 s	265.5 MB	18.01 s	545.7 MB
4.72 s	265.5 MB	36.77 s	725.5 MB
119.86 s	756.0 MB	64.13 s	804.7 MB
4.19 s	266.3 MB	42.56 s	878.3 MB

14×14			
PolyBoRi		Magma	
Time	Memory	Time	Memory
386.32 s	1659.5 MB	1238 s	6880.75 MB
11.85 s	279 MB	4962 s	14554.69 MB
9.32 s	275.3 MB	2111 s	10292.16 MB
401.83 s	2161.5 MB	903 s	4422.66 MB
823.32 s	3563.1 MB	1667 s	8539.16 MB

Table 17: *Comparison of CPU time and memory usage to solve binary puzzles using Gröbner bases algorithms for Boolean polynomials implemented in PolyBoRi and Magma.*

6×6			
Singular		Magma	
Time	Memory	Time	Memory
1.84 s	226.46 MB	0.46 s	35.4 MB
1.83 s	225.14 MB	0.46 s	38.3 MB
1.74 s	225.80 MB	0.27 s	28.9 MB
1.84 s	225.54 MB	0.63 s	38.9 MB
1.87 s	224.48 MB	0.45 s	33.6 MB

8×8			
Singular		Magma	
Time	Memory	Time	Memory
61.59 s	350.24 MB	61.84 s	2075.59 MB
5.32 s	245.12 MB	214.57 s	4033.69 MB
2.26 s	233.37 MB	4.57 s	152.12 MB
3.19 s	237.69 MB	12.19 s	488.4 MB
2.83 s	234.84 MB	5.72 s	158.35 MB

Table 18: *Comparison of CPU time and memory usage to solve binary puzzles as polynomials equations over $\mathbb{Q}$ using implementation of Gröbner bases in Singular and Magma.*

6×6			
Singular		Magma	
Time	Memory	Time	Memory
2.32 s	234.14 MB	0.38 s	43.84 MB
2.04 s	231.65 MB	0.54 s	46.43 MB
1.94 s	230.73 MB	0.24 s	30.07 MB
2.05 s	230.05 MB	0.41 s	44.64 MB
1.95 s	229.97 MB	0.22 s	35.88 MB

8×8			
Singular		Magma	
Time	Memory	Time	Memory
3.12 s	293.86 MB	778713 s	53762.45 MB
2.34 s	253.30 MB	4248.12 s	7486.61 MB
2.34 s	247.43 MB	88.04 s	203.57 MB
2.35 s	249.87 MB	336.85 s	1386.29 MB
3.98 s	266.30 MB	112.95 s	460.86 MB

Table 19: *Comparison of CPU time and memory usage to solve binary puzzles as polynomials equations over $\mathbb{Z}$ using implementation of Gröbner bases in Singular and Magma.*