Mathematics in Computer Science

Solving a Binary Puzzle

Putranto H. Utomo · Rusydi H. Makarim

Received: 7 December 2016 / Revised: 13 March 2017 / Accepted: 24 April 2017 / Published online: 10 May 2017

Abstract A Binary puzzle is a Sudoku-like puzzle with values in each cell taken from the set $\{0, 1\}$. Let $n \geq 4$ be an even integer, a solved binary puzzle is an $n \times n$ binary array that satisfies the following conditions: (1) no three consecutive ones and no three consecutive zeros in each row and each column; (2) the number of ones and zeros must be equal in each row and in each column; (3) there can be no repeated row and no repeated column. This paper proposes three approaches to solve the puzzle. The first method is based on a complete backtrack-based search algorithm. The idea is to propagate and fill an unsolved binary puzzle according to the three constraints, followed by a random guess if the puzzle remains unsolved. The second method of solving a binary puzzle is by representing it as an instance of a Boolean satisfiability problem which allows the solution for a binary puzzle to be obtained using SAT solvers. The third approach is based on expressing a binary puzzle as a system of polynomial equations over the binary field $\mathbb{F}_2$. The set of solutions for the equation system implies the solutions for the binary puzzle and it is obtained by computing a Gröbner basis of the ideal generated by the polynomials. We experimentally compare the three approaches with binary puzzles of various sizes and different numbers of empty cells using a computer algebra system.

Keywords Puzzle · NP-complete · Gröbner basis · SAT problem

Mathematics Subject Classification Primary 05A05; Secondary 12Y05

1 Introduction

A binary puzzle is a Sudoku-like puzzle with values in each cell taken from the set $\{0, 1\}$. The reader who is interested in the work related to Sudoku and mathematics can refer to [1,2,10,14]. We will now define the binary puzzle. Let $n \geq 4$ be an even integer. A solved binary puzzle is an $n \times n$ binary array that satisfies the following conditions/constraints:

P. H. Utomo (✉)
Eindhoven University of Technology, Eindhoven, The Netherlands
e-mail: p.h.utomo@tue.nl

R. H. Makarim
Mathematical Institute, University Leiden and Centrum Wiskunde en Informatica, Amsterdam, The Netherlands
e-mail: makarim@cwi.nl

	0						
			1		1		0
		0					
	1						
				1			
	0				1		
	0			0			
				0		0	

Fig. 1 Unsolved Puzzle

1	0	**1**	**0**	**1**	**0**	**1**	**0**
0	**1**	**0**	1	**0**	1	**1**	0
1	**0**	0	**1**	**1**	**0**	**0**	**1**
0	1	**1**	**0**	**0**	**1**	**1**	**0**
0	**1**	**0**	**1**	1	**0**	**0**	**1**
1	0	**1**	**0**	**1**	1	**0**	**0**
1	0	**0**	**1**	0	**0**	**1**	**1**
0	**1**	**1**	**0**	0	**1**	0	**1**

Fig. 2 Solved Puzzle

1. No three consecutive ones and also no three consecutive zeros in each row and each column,
2. every row and column is balanced, that is the number of ones and zeros must be equal in each row and in each column,
3. there can be no repeated row and no repeated column.

We refer to the above as the first, second, and third constraint of a binary puzzle respectively.

Figure 1 is an example of an initial setting of a binary puzzle. A solution that satisfies all three conditions can be seen in Fig. 2. It is possible that multiple solutions exist for a given binary puzzle. In the case of the puzzle in Fig. 1, there exists only one unique solution. In [7] de Biasi shows that deciding whether a binary puzzle has a valid solution is NP-complete. The proof is based on the reduction of deciding whether a planar conjunctive normal form is satisfiable, which is also an NP-complete problem [11]. De Biasi also proves that deciding the existence of a valid solution for another variant of a binary puzzle that omits the second and third constraints, called a *bare binary puzzle*, is NP-hard.

Utomo and Pellikaan studied binary puzzles as erasure decoding problem [19]. Their results provide the lower and upper bound for the rate of the codes, as well as the probability of correct erasure decoding for the puzzles. Another work by the same authors discusses binary puzzles as constrained arrays and presents some results on the rate of codes based on binary puzzles [20].

This work proposes different techniques in solving binary puzzles. We devise and compare three approaches to find its solution. The first approach solves binary puzzles as an array in the $\mathbb{F}_2^{n\times n}$ using a complete backtrack-based search algorithm. The idea is quite similar with the DPLL algorithm, which is finding values based on the three constraint, random guessing if necessary, and backtrack if there is a contradiction. The second approach is based on transforming a binary puzzle into a Boolean satisfiability (SAT) problem. And hence we have a different representation for the puzzle, which we solve later using a SAT solver. The third approach constructs a set of polynomial equations over the binary field representing the three conditions for a solved binary puzzle. The variables in the system of equations all correspond to cells in the puzzle. Hence the solution for the system of equations is a solution for the puzzle and it can be obtained by computing the Gröbner basis of the ideal generated by the polynomials in the equation system.

1.1 Notations

Let $n = 2m$ and $m \geq 2$. For an $n \times n$ binary puzzle and $i, j \in \{1, \ldots, n\}$ we use x_{ij} to denote Boolean variable representing the cell at row i and column j. Row 1 is the top-most row of the puzzle and column 1 is the left-most column. In the visual illustration of a binary puzzle, the cells with value 0 and 1 are represented in black and white-colored squares respectively while the gray-colored ones describe the cells with unknown value. The binary field is denoted by $\mathbb{F}_2 = \{0, 1\}$ and the n-dimensional vector space over $\mathbb{F}_2$ is denoted by $\mathbb{F}_2^n$. The Boolean operators OR, AND, and NOT are denoted by $\vee$, $\wedge$ and $\neg$ respectively. The notation for the binomial coefficient is written as $\binom{m}{k}$.

2 Backtrack-Based Search

One natural approach to reach a solution for a given unsolved binary puzzle is to follow the implications of the three constraints on some of its empty cells. The current state of the rows, columns, and some adjacent cells allow the deduction of value in the cells that are not yet determined. We discuss some of these implications for each binary puzzle constraint below.

1. *Constraint 1* Recall that the first constraint of binary puzzle is that no three neighbouring rows/columns have the same value. If it appears that the value of two out of three adjacent cells in a row/column were known and identical, then the value of the one remaining undetermined cell can be trivially deduced, which is equal to the negation of the known values. Examples of this condition in rows are illustrated in Fig. 3. Similarly, this is also applied for every three neighbouring column cells.
2. *Constraint 2* The second constraint states that the number of zeros and ones in every row and column must be equal. Equivalently as binary vector, each row/column of a binary puzzle is balanced. Following such a constraint, the deduction of cells with unknown values in a row or a column can be done when the number of zeros/ones is equal to $n/2$ (see Fig. 4).
3. *Constraint 3* The third constraint, that is no two rows/columns are equal, involves comparison of rows/columns in a binary puzzle. This allows the deduction of some value in cells of a row/column in the case when uniqueness can be guaranteed. Figure 5 illustrates an example when such a situation may occur.

The propagation of a binary puzzle is repeatedly applied until the puzzle is solved. However, it is very unlikely to obtain a valid solution for a binary puzzle directly only by following the implication of all three constraints. At a certain step, it is possible that no more values of empty cells can be deduced by the constraint propagation.

It is then necessary to perform random guesses on some empty cells. This is followed by another constraint propagation after some values are guessed. If the propagation yields a contradiction in fulfilling the three constraints,

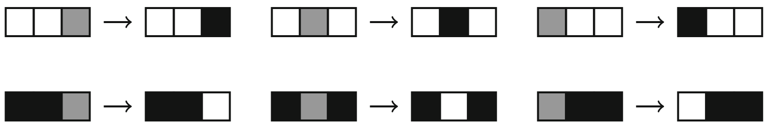

Fig. 3 Propagation in three-adjacent row cells implied by the first constraint

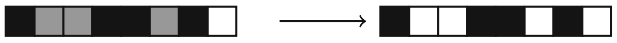

Fig. 4 An example of propagation implied by the second constraint in a row of length 8. Since the number of zeros is equal to 4, we can deduce that the remaining cells should be equal to 1

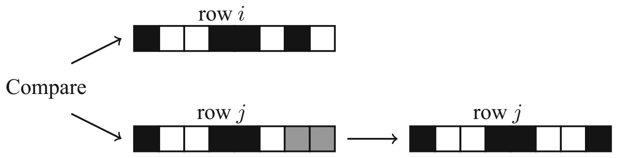

Fig. 5 An example of propagation implied by the third constraint. For $i \neq j$, the row i and j above are nearly equal except for the last two cells of row j that are initially unknown. The third constraint enforces these last two cells to have opposite value of their corresponding cells in row i

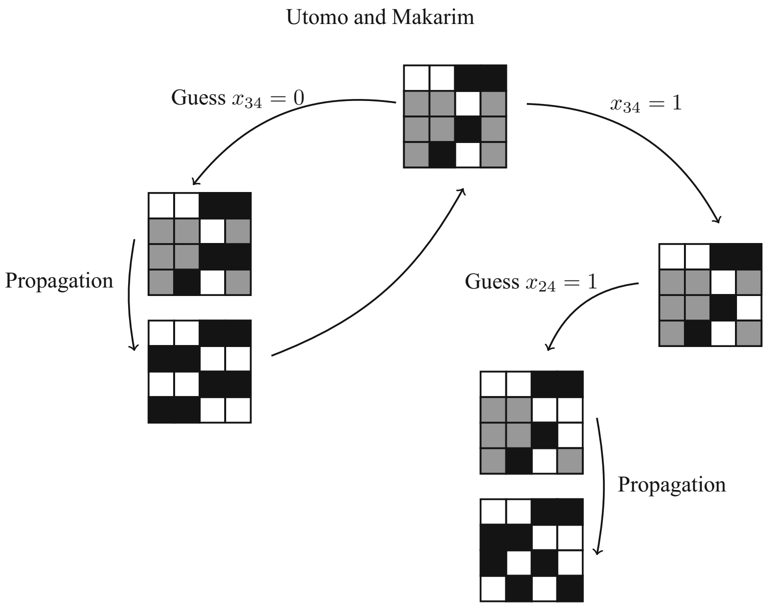

Fig. 6 Illustration of backtrack-based search in solving a binary puzzle. In this 4 × 4 binary puzzle, the top part indicates its initial state. After guessing $x_{34} = 0$, it shows that the third constraint is violated. Therefore the correct value of x_{34} must be equal to 1

then the guessed cells are clearly wrong. Thus, one should re-do and attempt a new constraint propagation with different values until no contradictions occur. Hence we need to keep track on all cells that has been guessed and propagated. We can also deduce that a random guess is never made if there is only one blank remaining. The entire search algorithm is described as a systematic backtracking search procedure. We illustrate this in Fig. 6. Here only one cell value is guessed each time a random-guess is performed. The pseudocode for the entire backtrack-based search to solve a binary puzzle is provided in Algorithms 1 and 2.

3 Binary Puzzles and the SAT Problem

Since each cell in the binary puzzle can only take the values '0' and '1', we can represent the puzzle as an array of binary variables, where false corresponds to '0' and true to '1'. We can express each condition in terms of a logical expression. We denote a variable or its negation as a literal and an expression formed from a finite collection of literals as a clause.

To illustrate the derivation of the first constraint, let x_1, x_2, x_3 be any three consecutive cells. There will be no three consecutive ones and zeros if and only if the following expression is true $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$. For satisfying the second constraint on balancedness on a vector with length $2m$, every $m+1$ cells must have at least one 1 and one 0. In other words, the following formula must be true for every possible way of selecting $m+1$ cells:

Algorithm 1: SOLVE(B, i, j, v)

```
Input: An $n \times n$ binary puzzle $B$, positive integers $i, j$ with $1 \le i, j \le n$, and $v \in \{0, 1\}$.
1  $B[i][j] \leftarrow v$
2  $B' \leftarrow$ ConstraintPropagation($B$)
3  if contradiction occur in $B'$ then
4      return
5  else
6      if $B'$ has no empty cell then
7          print $B'$
8          return
9      else
10         Randomly choose $x, y$ ($1 \le x, y \le n$) such that $B'[x][y]$ is an empty cell.
11         Solve ($B', x, y, 0$)
12         Solve ($B', x, y, 1$)
```

Algorithm 2: CONSTRAINTPROPAGATION(B)

```
Input: An $n \times n$ binary puzzle $B$.
Output: An updated $B$ according to all three constraints.
1  repeat
2      for $i = 1$ to $n$ do // Fulfilling Constraint 1
3          for $j = 1$ to $n$ do
4              if $j \le n - 2$ then
5                  if there is an empty cell in ($B[i][j]$, $B[i][j + 1]$, $B[i][j + 2]$) then
6                      Fill ($B[i][j]$, $B[i][j + 1]$, $B[i][j + 2]$) according to Constraint 1
7              if $i \le n - 2$ then
8                  if there is an empty cells in ($B[i][j]$, $B[i + 1][j]$, $B[i + 2][j]$) then
9                      Fill ($B[i][j]$, $B[i + 1][j]$, $B[i + 2][j]$) according to Constraint 1
10
11     for $i = 1$ to $n$ do // Fulfilling Constraint 2
12         if row $i$ of $B$ has an empty cell then
13             Fill row $i$ of $B$ according to Constraint 2
14         if column $i$ of $B$ has an empty cell then
15             Fill column $i$ of $B$ according to Constraint 2
16
17     for $i = 1$ to $n - 1$ do // Fulfilling Constraint 3
18         if row $i$ of $B$ has no empty cell then
19             for $j = i + 1$ to $n$ do
20                 if row $j$ of $B$ has an empty cell then
21                     Fill row $j$ of $B$ according to Constraint 3 by comparing it with row $i$
22         if column $i$ of $B$ has no empty cell then
23             for $j = i + 1$ to $n$ do
24                 if column $j$ of $B$ has an empty cell then
25                     Fill column $j$ of $B$ according to Constraint 3 by comparing it with column $i$
26 until $B$ can no longer be updated
27 return $B$
```

$\left(\bigvee_{k=1}^{m+1} x_k\right) \wedge \left(\bigvee_{k=1}^{m+1} \neg x_k\right)$. The last constraint is straightforward. Suppose we have two vectors **x** and **y** with length $2m$. Since **x** must be different to **y**, then the following formula must be satisfied: $\neg \bigwedge_{i=1}^{2m} [(x_i \wedge y_i) \vee (\neg x_i \wedge \neg y_i)]$.

Now, suppose we have an $n \times n$ array in the variables x_{ij}. The array satisfies the first condition, that there are no three consecutive ones and also no three consecutive zeros in each row and each column, if and only if the expression below is true:

$$\left(\bigwedge_{i=1}^{2m} \left\{ \bigwedge_{k=1}^{2m-2} \left(\left[\bigvee_{j=k}^{k+2} x_{ij} \right] \wedge \left[\bigvee_{j=k}^{k+2} \neg x_{ij} \right] \right) \right\} \right) \wedge$$
$$\left(\bigwedge_{j=1}^{2m} \left\{ \bigwedge_{k=1}^{2m-2} \left(\left[\bigvee_{i=k}^{k+2} x_{ij} \right] \wedge \left[\bigvee_{i=k}^{k+2} \neg x_{ij} \right] \right) \right\} \right).$$

For satisfying the second condition on balancedness, the following expression must be true

$$\left(\bigwedge_{j=1}^{2m} \left[\bigwedge_{1 \le i_1 < \cdots < i_{m+1} \le 2m} \left(\bigvee_{k=1}^{m+1} x_{i_k j} \right) \right] \right) \wedge \left(\bigwedge_{i=1}^{2m} \left[\bigwedge_{1 \le j_1 < \cdots < j_{m+1} \le 2m} \left(\bigvee_{k=1}^{m+1} x_{i j_k} \right) \right] \right) \wedge$$
$$\left(\bigwedge_{j=1}^{2m} \left[\bigwedge_{1 \le i_1 < \cdots < i_{m+1} \le 2m} \left(\bigvee_{k=1}^{m+1} \neg x_{i_k j} \right) \right] \right) \wedge \left(\bigwedge_{i=1}^{2m} \left[\bigwedge_{1 \le j_1 < \cdots < j_{m+1} \le 2m} \left(\bigvee_{k=1}^{m+1} \neg x_{i j_k} \right) \right] \right).$$

Since for each row or column we need to check every $m + 1$ cells from $2m$ cells for both symbol 0 and 1, the complexity of this expression grows as $4 \cdot 2m\binom{2m}{m+1}$ which is exponential in m.

The satisfiability of the third condition, that every two rows and every two columns must be distinct, is equal to

$$\left(\bigwedge_{1 \le j_1 < j_2 \le 2m} \left\{ \neg \bigwedge_{i=1}^{2m} \left[\left(x_{ij_1} \wedge x_{ij_2} \right) \vee \left(\neg x_{ij_1} \wedge \neg x_{ij_2} \right) \right] \right\} \right) \wedge$$
$$\left(\bigwedge_{1 \le i_1 < i_2 \le 2m} \left\{ \neg \bigwedge_{j=1}^{2m} \left[\left(x_{i_1 j} \wedge x_{i_2 j} \right) \vee \left(\neg x_{i_1 j} \wedge \neg x_{i_2 j} \right) \right] \right\} \right).$$

Since many SAT solving algorithms often assume that the proposition is in CNF (Conjunctive Normal Form), it is necessary to convert our expression into CNF. However, conversion to CNF can lead to an exponential explosion of the formula.

We say a formula in the CNF if it is a conjunction of clauses, where a clause is a disjunction of literals. For the first constraint, the formula is already in CNF. Moreover, each three consecutive cells has two clauses having three literals in each clause. Since there are $4m$ vector and every vector has $2m - 2$ three consecutive cells, we have $2 \cdot 4m(2m - 2)$ clauses, and each clause has 3 literals.

Recall the derivation of second constraint. In checking a vector, there are $2\binom{2m}{m+1}$ clauses. Hence in total, there are $4m \cdot 2\binom{2m}{m+1}$ clauses, where each clause has $m + 1$ literals.

The easiest way to make CNF from the third constraint is by using the following procedure. Suppose we want to check the i^{th} and j^{th} column, **x** and **y** respectively. Then we can enumerate all 2^{2m} combinations of $\bigvee_{i=1}^{2m}(x_i \vee y_i)$. Hence we have $2 \cdot 2^{2m}\binom{2m}{2}$ clauses where each clause has $4m$ literals for the third constraint. One can see that this will blow up exponentially in terms of m. One way to overcome this is by using Tseytin transformation [13,16]. The idea of Tseytin transformation is to make a CNF from any Boolean formula by introducing some fresh variables representing a clause subformula. With this approach, the transformation will output a formula whose size is linear in terms of the input formula. For a 2-variable x, y and OR operator with the Tseytin variable z, we will have the following CNF, which have 3 clauses.

$(\neg z \vee x \vee y) \wedge (z \vee \neg x) \wedge (z \vee \neg y)$.

We will illustrate the Tseytin transformation of third constraint only for the columns, that is $\left(\bigwedge_{1\le j_1<j_2\le 2m} \left\{\neg \bigwedge_{i=1}^{2m} \left[\left(x_{ij_1} \wedge x_{ij_2}\right) \vee \left(\neg x_{ij_1} \wedge \neg x_{ij_2}\right)\right]\right\}\right)$. Since the outer operator is already in conjunction, we only need to transform the inner part, $\left\{\neg \bigwedge_{i=1}^{2m} \left[\left(x_{ij_1} \wedge x_{ij_2}\right) \vee \left(\neg x_{ij_1} \wedge \neg x_{ij_2}\right)\right]\right\}$. We define the Tseytin variable a_i, b_i, c_i for $i = 1, \ldots, 2m$, and d as follow.

$$\begin{aligned}
a_i &= x_{ij_1} \wedge x_{ij_2}\\
b_i &= \neg x_{ij_1} \wedge \neg x_{ij_2}\\
c_i &= a_i \vee b_i\\
d &= \neg \bigwedge_{i=1}^{m} c_i
\end{aligned}$$

Each transformation for the equations having a_i, b_i, and c_i as the Tseytin variable will produce three clauses for each i. Enumerating for all transformations in a_i, b_i, and c_i for $i = 1, \ldots, 2m$, we have $3 \cdot 3 \cdot 2m$ clauses. Meanwhile, the CNF of $d = \neg \bigwedge_{i=1}^{m} c_i$ is $\left(\neg d \vee \bigvee_{i=1}^{2m} \neg c_i\right) \wedge \left(\bigwedge_{i=1}^{2m} (c_i \vee d)\right)$, and it has $2m+1$ clauses. Therefore, we have $10 \cdot 2m + 1$ clause for the transformation of the inner part. Hence CNF transformation of third constraint will have $2\binom{2m}{2}(10 \cdot 2m + 1)$ clauses and each clause will have at most $2m + 1$ literals.

Once in CNF, we use a general SAT solver to find a solution for the puzzle, that is CryptoMiniSat [15].

4 Binary Puzzles and Systems of Polynomial Equations

This section discusses how a problem of solving binary puzzles can be seen as equivalent to finding solutions to a system of multivariate polynomial equations over $\mathbb{F}_2$. We first recall some notions in the theory of Boolean functions that are relevant for this section.

The Hamming weight of $u \in \mathbb{F}_2^n$, that is the number of nonzero components of u, is denoted by $\mathrm{wt}(u)$. Let $v, w \in \mathbb{F}_2^n$ where $v = (v_1, \ldots, v_n)$, $w = (w_1, \ldots, w_n)$. We say that v is covered by w (or w covers v), denoted by $v \preceq w$, if $v_i \le w_i$ for all $i \in \{1, \ldots, n\}$. Similarly, let $p, q \in \mathbb{Z}_{\ge 0}$ be non-negative integers, we say that $p \preceq q$ if the binary representation of p is covered by the binary representation of q.

An n-variable Boolean function f is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. A natural way to represent f is by listing $f(x_1, \ldots, x_n)$ for every $(x_1, \ldots, x_n) \in \mathbb{F}_2^n$ ordered lexicographically. This representation is called the *truth table* of f. The function f can also be represented using a multivariate polynomial in the following form

$$\sum_{u\in\mathbb{F}_2^n} a_u x_1^{u_1} \cdots x_n^{u_n} \tag{4.1}$$

where $u = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$ and $a_u \in \mathbb{F}_2$. The representation in (4.1) is called the *algebraic normal form* (ANF) of f. The conversion of a Boolean function from its truth table to the corresponding algebraic normal form is done using the following result [5].

Theorem 4.1 *Let f be an n-variable Boolean function and let $\sum_{u\in\mathbb{F}_2^n} a_u x_1^{u_1} \cdots x_n^{u_n}$ be its ANF. For all $u \in \mathbb{F}_2^n$ we have*

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

A system of polynomial equations that represent an $n \times n$ binary puzzle consists of polynomials with indeterminates x_{ij} for all $i, j \in \{1, \dots, n\}$. The indeterminates represent the value at row i and column j and the polynomials exhibit all three conditions/constraints of a solved binary puzzle. In the following we show how the three constraints are represented as multivariate polynomials.

Theorem 4.2 *Let x, y, z be variables that represent three adjacent cells in a row (or column) of an $n \times n$ binary puzzle. The first constraint of a binary puzzle is represented by the following polynomial equation*

$$f_1(x, y, z) = xy + xz + x + yz + y + z + 1. \tag{4.2}$$

Proof Recall that the first constraint of a binary puzzle is there should not be any three consecutive zeros and ones in each row and column. Let f_1 be the polynomial equation that represent such constraint and by definition f_1 must satisfy the following

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

By Theorem 4.1, $a_u = 0$ only when $u = (1, 1, 1)$. Thus $f_1(x, y, z) = xy + xz + x + yz + y + z + 1$. □

Lemma 4.3 *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.*

Proof See [12]. □

Theorem 4.4 *Let $x_1, \dots, x_n$ be variables that denote the n cells of a single row/column of an $n \times n$ binary puzzle. The polynomial equation that represents the second constraint is given by*

$$f_2(x_1, \dots, x_n) = \sum_{\substack{u \in \mathbb{F}_2^n \setminus \{0\} \\ n/2 \preceq \mathrm{wt}(u)}} x_1^{u_1} \cdots x_n^{u_n} + 1. \tag{4.3}$$

Proof By definition, the function $f_2 : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is defined by

$$f_2(x_1, \dots, x_n) = \begin{cases} 0 & \text{if } \mathrm{wt}(x_1, \dots, x_n) = n/2 \\ 1 & \text{otherwise.} \end{cases} \tag{4.4}$$

By Theorem 4.1 the coefficient a_0 in the ANF of f_2 is $a_0 = f_2(0) = 1$. To compute the remaining coefficient a_u for nonzero $u \in \mathbb{F}_2^n$, we will divide it into two cases for $\mathrm{wt}(u) < n/2$ and $\mathrm{wt}(u) \geq n/2$.

For all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) < n/2$, by Theorem 4.1 we have $a_u = \sum_{w \preceq u} f_2(w) = 0$ because $f_2(w) = 1$ for all $w \preceq u$ and the cardinality of $\{w \in \mathbb{F}_2^n \mid w \preceq u\}$ is even.

For all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) \geq n/2$: by (4.4), $f_2(w) = 0$ for $w \preceq u$ if and only if $\mathrm{wt}(w) = n/2$. Let $\mathrm{Supp}(u) = \{i \mid u_i \neq 0\}$ denotes the support of u. The number of w, $w \preceq u$ such that $\mathrm{wt}(w) = n/2$ is equal to $\binom{|\mathrm{Supp}(u)|}{n/2} = \binom{\mathrm{wt}(u)}{n/2}$. Thus for all nonzero $u \in \mathbb{F}_2^n$ such that $\mathrm{wt}(u) \geq n/2$ the value $a_u = \sum_{w \preceq u} f_2(w) = 1$ 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 Lemma 4.3. This completes the proof. □

Theorem 4.5 *Let $x_1, \dots, x_n$ and $y_1, \dots, y_n$ be variables that represent two distinct rows (or columns) in an $n \times n$ binary puzzle. The polynomial equation that represent the third constraint of a binary puzzle is given by*

$$f_3(x_1, \dots, x_n, y_1, \dots, y_n) = \prod_{i=1}^{n} (x_i + y_i + 1). \tag{4.5}$$

Proof The third constraint is satisfied if there exists $i \in \{1, \ldots, n\}$ such that $x_i + y_i = 1$. Polynomial (4.5) is then a trivial implication from the definition of the third constraint. □

The size of an equation system corresponding to a $n \times n$ binary puzzle in terms of number of polynomials can be conveniently derived. In a single row/column, the number of polynomials (4.2) that represent the first constraint of an $n \times n$ binary puzzle is $n - 2$. Since we have n rows and n columns, there are $2n(n-2) = 2n^2 - 4n$ polynomials of the form (4.2) in the equation system. The number of polynomials of the form (4.3) for the second constraint is clearly equal to $2n$. For two distinct rows/columns, we obtain one polynomial of the form (4.5). Thus, the number of polynomials that represent the third constraint is equal to $2(n(n-1)/2) = n(n-1)$. In total, the equation system that represents all three constraints of an $n \times n$ binary puzzle consists of $2n^2 - 4n + 2n + n^2 - n = 3n(n-1)$ polynomials in n^2 variables. The degree of the equation system is equal to n due to polynomial (4.5) and the fact that the degree of (4.3) can not be strictly larger than n.

4.1 Solving Binary Puzzles Using Gröbner Bases

Once an equation system is constructed for a binary puzzle, the solutions of the system implies the solution for the binary puzzle as well. One way to obtain the solutions for a system of polynomial equations is by computing the Gröbner basis of the ideal generated by these polynomials.

Let $\mathcal{R}$ be a polynomial ring in n variables over a field $\mathbb{F}$. We adapt an admissible ordering in the set of all monomials in $\mathcal{R}$. Let $f \in \mathcal{R}$, we denote by $\mathrm{LM}(f)$ the largest monomial appearing in f.

Definition 4.6 (*Gröbner bases*) Let $F = \{f_1, \ldots, f_m\} \subset \mathcal{R}$ and let $\mathcal{I}$ be an ideal in $\mathcal{R}$ generated by polynomials in F. A finite subset $G \subset \mathcal{I}$ that generates $\mathcal{I}$ is said to be a Gröbner basis of $\mathcal{I}$ if for any $f \in \mathcal{I}$ there exists $g \in G$ such that $\mathrm{LM}(g)$ divides $\mathrm{LM}(f)$.

Historically, the method to compute Gröbner basis of a polynomial ideal was introduced by Buchberger [4]. Since then, various improvements have been proposed in the literature, notably with the seminal work of Faugère's F_4 [8] and F_5 [9] algorithms. These two algorithms employ linear algebra techniques to perform multiple polynomial reduction at once.

The choice of monomial ordering in a Gröbner basis computation has an impact in the shape of Gröbner basis itself as well as the time complexity of the computation. For example, a Gröbner basis of a polynomial ideal corresponding to lexicographic monomial ordering has the following form :

$$\{g_1(x_1), \ldots, g_2(x_1, x_2), \ldots, g_{k_1}(x_1, x_2), g_{k_1+1}(x_1, x_2, x_3), \ldots, g_{k_n}(x_1, \ldots, x_n)\}.$$

With the above structure, recovering the set of solutions is done by successively eliminating variables. One begin by factoring the univariate polynomials and performs back-substitution to other polynomials. A similar process is then repeatedly applied for new univariate polynomials produced after back-substitution, until all the solutions are obtained.

Note that in the case of binary puzzles we are only interested with solutions that lie in the base field $\mathbb{F}_2$. There are two practices to impose such restriction, which will allow us to read the solution directly from the Gröbner basis. The first method is to add the *field equation* polynomials $x_{ij}^2 + x_{ij}$ for all $i, j \in \{1, \ldots, n\}$ to the equation system, adding n^2 more polynomials. For a general finite field $\mathbb{F}_q$ this approach is beneficial whenever q is less than or equal than the degree of the system. The second method is to define the polynomials over the quotient ring $\mathcal{R}/\mathcal{I}$ instead of $\mathcal{R}$ where $\mathcal{I}$ is the ideal generated by the polynomials $x_{ij}^2 + x_{ij}$ for all $i, j \in \{1, \ldots, n\}$. This is considered as a more natural practice when dealing with Boolean polynomial algebraically. Furthermore, dedicated implementations for computation of Boolean polynomials are available in existing computer algebra software such as Magma [3] and SageMath [17]. These implementations reduce significantly amounts of memory usage while at the same time provide an excellent speedup due to its specialized data structure compared to the general implementation of multivariate polynomial.

Table 1 Comparison of execution time (in seconds) for each method

Size	SAT		SAT (with Tseytin tranf.)		Gröbner basis[a]		Backtrack-based search
	Pre-comp.	Solver	Pre-comp.	Solver	Pre-comp.	Solver	
4×4	0.02	0.00	0.18	0.001	0.02	0.05	0.03
6×6	0.14	0.02	0.66	0.003	0.11	0.06	0.19
8×8	1.45	0.10	1.59	0.01	0.53	0.13	0.49
10×10	10.80	0.69	3.24	0.02	3.30	8.69	5.26
12×12	81.87	4.67	6.09	0.08	47.80	4.55	8.05
14×14	–	–	10.74	0.32	–	–	78.67
16×16	–	–	19.42	1.43	–	–	974.58
18×18	–	–	45.33	6.56	–	–	65445.50

[a] All puzzles used in this experiment have a unique solution

In addition to strategies above that eliminate irrelevant solutions, practically one would also intend to obtain the solutions in an efficient manner. A critical parameter that influences the performance of Gröbner bases computation is the choice of monomial ordering. Even though using lexicographic ordering allows to "read" the set of solutions to a system of multivariate polynomial equations, however computationally it is considered as a less efficient monomial ordering compared to orderings that respect the total degree of a monomial such as degree lexicographic or degree-reverse lexicographic ordering. For more detailed treatment on the theory of Gröbner basis the reader may refer to [6].

The polynomial equations that represent an $n \times n$ binary puzzle need to be constructed only once. For different binary puzzle of the same size, we can substitute variables x_{ij} with $c_{ij} \in \mathbb{F}_2$ where c_{ij} is the initial value of entry at row i and column j. Equivalently, we may also add linear polynomials $x_{ij} + c_{ij}$ in the equation system.

5 Experimental Results

Simulation has been done under 64 bit Debian 8, with hardware specification: Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz, 10GB RAM. The software used for the experiment is SageMath 7.2 [17] as the wrapper and front-end programming language. Reader who interested in the experiment can refer to [18] for the source code including the input and output of our simulation.

In this section, we compare the three methods for solving binary puzzles. In the experiment, a puzzle is represented in a form of a matrix with entries $\{0, 1, 9\}$, where 9 represents a blank. We generate 6 different binary puzzles for each size. To be used in the experiment, we randomly select cells to be erased, hence they might have more than one solution. The result in the Table 1 and both in the Figs. 7 and 8 are acquired by taking the average of 18 experiments.

Since the SAT solver and the Gröbner basis solver require different types of input to work with, we need to do pre-computations for those methods, that is transforming the initial form of the puzzle into the form required by the solver. For the SAT method, creating a CNF for the puzzle will be the precomputation. Similarly, creating the Boolean function for the puzzle will be the precomputation for the Gröbner basis method.

In this experiment, we use CryptoMiniSat 2 for the SAT solver and BRiAl[1] 0.8.4.3 for the computation of Gröbner bases. Both packages and their interfaces are available in SageMath 7.2. Note that Gröbner bases computation in the following experiment is done in the quotient ring $\mathcal{R}/\mathcal{I}$ with degree lexicographic monomial ordering.[2] For backtrack-based search, we implement the algorithm for solving the binary puzzle in SageMath 7.2 [17]. We have several remarks:

- From the experiment, solving the puzzle by a SAT solver with Tseytin transformation is more efficient in terms of execution time.

[1] Previously known as PolyBori—http://polybori.sourceforge.net/. The source code is available at https://github.com/BRiAl/BRiAl.

[2] Even though using degree-reverse lexicographic ordering is practically considered to be more efficient to compute Gröbner bases, such ordering is not supported by BRiAl.

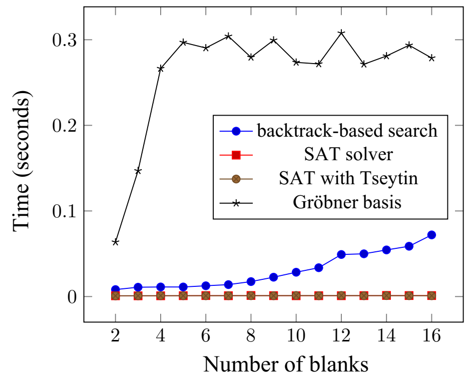

Fig. 7 Time comparison for different number of blanks in 4×4 puzzle

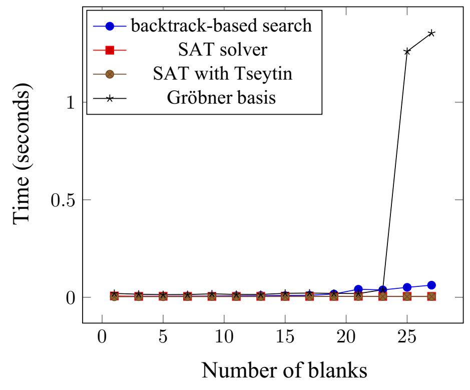

Fig. 8 Time comparison for different number of blanks in 6×6 puzzle

- Unlike backtrack-based search and Gröbner basis, the number of blanks hardly affect the SAT solver running time.[3] See Figs. 7 and 8 for 4×4 and 6×6 puzzle, respectively.[4]
- The SAT and Gröbner basis methods are limited by the memory aspect. With the current hardware specification, 18×18 is the limit. This is also explain the dash in the Table 1.
- Since the Gröbner basis method aims to find all solution(s) of the puzzle, it often fails to solve a puzzle having more than one solution.

The comparison between the three methods in solving a particular puzzle of various sizes is given in the Table 1. In these puzzles, around 75% of the cells were blanks.

References

1. Arnold, E., Lucas, S., Taalman, L.: Gröbner basis representations of sudoku. Coll. Math. J. **41**(2), 101–112 (2010)

[3] The timing in these plots do not include the prepossessing.

[4] 6×6 is the highest size such that the Gröbner basis method could find the solutions for most number of blanks.

2. Bailey, R.A., Cameron, P.J., Connelly, R.: Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and hamming codes. Am. Math. Mon. **115**, 383–404 (2008)
3. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symb. Comput. **24**(3–4), 235–265 (1997). Computational algebra and number theory (London, 1993)
4. Buchberger, B.: Bruno buchbergers phd thesis 1965: an algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. J. Symb. Comput. **41**(3), 475–511 (2006)
5. Carlet, C.: Boolean functions for cryptography and error correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 257–397. Cambridge University Press, Cambridge (2010)
6. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms—An Introduction to Computational Algebraic Geometry and Commutative Algebra. Undergraduate Texts in Mathematics, 4th edn. Springer, Berlin (2015)
7. de Biasi, M.: Binary Puzzle is NP-Complete. http://www.nearly42.org/vdisk/cstheory/binaryp.pdf (July 2012)
8. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases (F4). J. Pure Appl. Algebra **139**(1–3), 61–88 (1999)
9. Faugère, J.-C.: A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation. ISSAC '02, pp. 75–83. NY, USA, ACM, New York (2002)
10. Felgenhauer, B., Jarvis, F.: Mathematics of sudoku i, ii. Math. Spectr. **39**(15–22), 54–58 (2006)
11. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11**(2), 329–343 (1982)
12. Lucas, E.: Théorie des fonctions numériques simplement périodiques. Am. J. Math. **1**(2), 184–196 (1878)
13. Seltner, H.: Extracting hardware circuits from CNF formulas. Master's Thesis, Institute of Formal Models and Verification, Johannes Kepler Universität Linz (2014). http://fmv.jku.at/master/Seltner-MastherThesis-2014.pdf
14. Soedarmadji, E., Mceliece, R. J.: Iterative decoding for sudoku and latin square codes. http://leecenter.caltech.edu/workshop08/papers/mceliece2.pdf
15. Soos, M.: The CryptoMiniSat 5 set of solvers at SAT competition 2016. In: Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions—SAT Competition 2016, p. 28 (2016). https://github.com/msoos/cryptominisat
16. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483. Springer, Berlin (1983)
17. SageMath, the Sage Mathematics Software System (Version 7.2), The Sage Developers (2016). http://www.sagemath.org
18. Utomo, P., Makarim, R.: The solver of binary puzzle. https://github.com/puth7/binarysolver
19. Utomo, P., Pellikaan, R.: Binary puzzles as an erasure decoding problem. In: Jérémie, R., Francois H. (eds.) Proceedings of the 36th WIC Symposium on Information Theory in the Benelux, pp. 129–134 (May 2015). http://www.win.tue.nl/~ruudp/paper/72.pdf
20. Utomo, P., Pellikaan, R.: On the rate of constrained arrays. In: Proceedings IndoMS International Conference on Mathematics and its Applications (IICMA) (November 2015). http://www.win.tue.nl/~ruudp/paper/75.pdf