Binary puzzle as a SAT problem

Putranto Utomo[1,2] Ruud Pellikaan[1]
p.h.utomo@tue.nl g.r.pellikaan@tue.nl
[1]Eindhoven University of Technology
Dept. Mathematics and Computer Science
P.O. Box 513. 5600 MB Eindhoven
[2]Sebelas Maret University
Fac. of Mathematics and Natural Sciences

In: Prodeedings of the 2017 Symposium on Information Theory and Signal Processing in the Benelux, May 11-12, 2017, Delft, the Netherlands
R. Heusdens and J. H. Weber (Eds.), pp. 223-229 , 2017

Abstract

A binary puzzle is a Sudoku-like puzzle consisting of a square array where each entry consists of a zero, a one, or a blank. 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 are no repeated rows and no repeated columns. We outline several mathematical problems related to the binary puzzle in [4]. We refer to [2] for further results on the rate of constrained arrays based on the binary puzzle.

This paper discusses the Boolean satisfiability (SAT) problem related to binary puzzles. A binary puzzle can be represented as an instance of a SAT problem. Each of the three constraints above is expressed by its equivalent logical expression. De Biasi showed that this puzzle is NP-complete [5]. We notice that there is one main drawback using this technique: the exponential blowing up in the conversion, both in the conversion to a SAT problem and also in the conversion from the SAT problem into a CNF [3]. The aim of this research is to create an algorithm that will result in a polynomial size CNF in terms of the size of the puzzle. In doing so, we apply the Tseytin transformation and a recursive formula for the second constraint.

1 Binary puzzle

The game of Binary puzzle is a one person board game where the player has to put a bit 1 or 0 in a partially filled array such that

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. No repeated rows and no repeated columns are allowed.

The constraints above imply that the number of rows and the number of columns are even, and we will consider only the case that these numbers are the same. An example of an initial puzzle can be seen in the Figure 1 and the solution is given in the Figure 2.

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

Figure 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**

Figure 2: Solved Puzzle

2 Binary puzzle as SAT problem

Let Γ be a logical formula, that is a grammatical correct expression in a set of binary variables, the constants 0 and 1, and the operations $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and $\oplus$ (exclusive or). The satisfiability (SAT) problem is the problem to find an assignment to all variables such that Γ is true. We call a literal a binary variable or its negation and a clause a disjunction of a finite set of literals. If Γ consists only of a conjunction of clauses, we say that Γ is in the Conjunctive Normal Form (CNF).

Recall that each cell in the solved binary puzzle can only take the values '0' and '1'. Hence 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. After representing the puzzle in the form of a expression, we use the SAT solver to solve the puzzle. But, since many SAT solving algorithms often assume that the proposition is in the CNF, it is necessary to convert our expression into the CNF. However, conversion to the CNF can lead to an exponential blow up of the formula. It is also known that the complexity of converting a formula into the CNF in general is NP-hard [8].

2.1 The 1^{st} constraint

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 $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

Now, suppose we have an $2m \times 2m$ 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:

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

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

This formula of the first constraint is already in CNF. Moreover, each three consecutive cells have two clauses with three literals in each clause. Since there are $4m$ vector

and every vector has $2m-2$ three consecutive cells, we have $8m(2m-2)$ clauses, and each clause has 3 literals.

2.2 The 2^{nd} constraint

For satisfying the second constraint, that is the balancedness of a vector of 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:

$$\left(\bigvee_{k=1}^{m+1} x_k\right) \wedge \left(\bigvee_{k=1}^{m+1} \neg x_k\right).$$

Hence, for a whole puzzle, 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 symbols 0 and 1, the complexity of this expression grows as $8m\binom{2m}{m+1}$ which is exponential in m. This also implies that there are $8m\binom{2m}{m+1}$ clauses, where each clause has $m+1$ literals.

An alternative polynomial expression for the satisfiability of the second constraint can be obtained as follows: Let $\mathbf{x} = (x_1, \ldots, x_{2m})$ be a row or column of the puzzle and let $\mathbf{y} = (y_1, \ldots, y_p)$ be the binary representation of the weight of $\mathbf{x}$, that is $\mathrm{wt}(\mathbf{x}) = \sum_{i=1}^{p} y_i 2^{i-1}$. Let $\mathbf{b} = (b_1, \ldots, b_l)$ be the binary representation of m. So $m = \sum_{i=1}^{l} b_i 2^{i-1}$. The second constraint is satisfied if $p = l$ and $b_i = y_i$ for all i .

Our next step is to find a polynomial algorithm to compute the weight of $\mathbf{x}$. The idea is that we iteratively sum the vector $\mathbf{x}$ from length 1 up to length n. In doing so, we need a temporary variable for storing the memory of the summation. Moreover, since we are working with formulas and variables, we need to store all the memory.

We will illustrate this by doing the addition of two integers m and n in 2-ary representation, $\mathbf{a} = (a_1, \ldots, a_l)$ and $\mathbf{b} = (a_1, \ldots, b_l)$ respectively, where $m = \sum_{i=1}^{l} a_i 2^{i-1}$, and $n = \sum_{i=1}^{l} b_i 2^{i-1}$. Then let $\mathbf{c} = (c_1, \ldots, c_l)$ be the representation of $m+n$, that means $m+n = \sum_{i=1}^{l} c_i 2^{i-1}$. Here we assume that m and n have the same length of the binary representation and moreover that $a_l = b_l = 0$. Let $\mathbf{t} = (t_1, \ldots, t_l)$ be the memory variable for calculating $\mathbf{c}$, such that $t_1 = 0$ and $t_{i+1} = (a_i \wedge b_i) \oplus (a_i \wedge t_i) \oplus (b_i \wedge t_i)$. Then $c_i = a_i \oplus b_i \oplus t_i$.

In our case, since we sum a vector recursively, it is easier to store the calculation in a matrix. Let $n = 2m$ and $l = \lceil \log_2(n+1) \rceil$. Let Z be an $n \times l$ matrix with entries z_{ij} and let T be an $n \times l$ matrix with entries t_{ij}. Here T serves as a memory variable and the i-th row of Z is the binary representation of the weight of $(x_1, \ldots, x_i)$. Hence the last row of Z is the binary representation of $\mathrm{wt}(\mathbf{x})$.

Then the definitions of z_{ij} and t_{ij} are given as follows.

$$z_{ij} = \begin{cases} x_1 & \text{if } i = j = 1 \\ z_{i-1,1} \oplus x_i & \text{if } i > 1 \text{ and } j = 1 \\ z_{i-1,j} \oplus t_{ij} & \text{if } i > 1 \text{ and } j > 1 \\ 0 & \text{elsewhere,} \end{cases}$$

$$t_{ij} = \begin{cases} 0 & \text{if } i = 1 \text{ or } j = 1 \\ x_i \wedge z_{i-1,1} & \text{if } i > 1 \text{ and } j = 2 \\ t_{i,j-1} \wedge z_{i-1,j-1} & \text{if } i > 1 \text{ and } j > 2. \end{cases}$$

Hence, the second constraint is satisfied if the following equation $z_{nj} = b_j$ is satisfied for all j and for all columns and for all rows.

One drawback of this construction is that the formula is not in CNF. We proceed with the Tseytin transformation since it will give an exponential blow up if we convert it using the logical equivalence rules, such as the double negation law, De Morgan's law, and the distributive law. The idea of Tseytin transformation is to make a CNF from Boolean formula by introducing some fresh variables representing a clause subformula [9, 7]. With this approach, the transformation will output a formula whose size is linear in terms of the input formula. For example, the 2-variable x, y and OR operator with the Tseytin variable z gives the following CNF, which has 3 clauses.

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

For simplicity, we only show the transformation of the first and second part of z_{ij}. Since $z_{ij} = x_1$ for $i = j = 1$, we will get $(\neg z_{11} \vee x_1) \wedge (z_{11} \vee \neg x_1)$ for the first part. For the second part, that is $z_{i1} = z_{i-1,1} \oplus x_i$, will be transformed into

$$(\neg z_{i1} \vee \neg z_{i-1,1} \vee \neg x_i) \wedge (z_{i1} \vee z_{i-1,1} \vee \neg x_i) \wedge (z_{i1} \vee \neg z_{i-1,1} \wedge x_i) \wedge (\neg z_{i1} \vee z_{i-1,1} \vee x_i)\,.$$

Now, we determine the size of the formula in CNF. Each row and each column will produce two matrices Z and T of size $n \times l$. From the definition of z_{ij}, since the most expensive transformation is $\oplus$, each cell will have at most 4 clauses and each clause will have at most 3 literals. From the definition of t_{ij}, each cell will have at most 3 clauses and each clause will have at most 3 literals. Hence for a single vector, we have at most $7nl$ clauses where each clause has at most 3 literals. Enumerating for all columns and rows, we will have at most $14ln^2$ clauses where each clause has at most 3 literals. So this recursive construction has polynomial complexity in space.

2.3 The 3^{rd} constraint

The last constraint is straightforward. Suppose we have two vector $\mathbf{x}$ and $\mathbf{y}$ with length $2m$. Since $\mathbf{x}$ must be different with $\mathbf{y}$, then the following formula must be satisfied:

$$\neg \bigwedge_{i=1}^{2m} \left[(x_i \wedge y_i) \vee (\neg x_i \wedge \neg y_i)\right].$$

The satisfiability of the third condition, that every two rows and every two columns

must be distinct, is given by

$$\left(\bigwedge_{1\le j_1<j_2\le 2m}\left\{\neg\bigwedge_{i=1}^{2m}\left[(x_{ij_1}\wedge x_{ij_2})\vee(\neg x_{ij_1}\wedge\neg x_{ij_2})\right]\right\}\right)\wedge$$
$$\left(\bigwedge_{1\le i_1<i_2\le 2m}\left\{\neg\bigwedge_{j=1}^{2m}\left[(x_{i_1j}\wedge x_{i_2j})\vee(\neg x_{i_1j}\wedge\neg x_{i_2j})\right]\right\}\right).$$

The easiest way to make a CNF from this expression is by using the following procedure. Suppose we want to check the i^{th} and j^{th} column, $\mathbf{x}$ and $\mathbf{y}$ 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 the Tseytin transformation which is briefly explained in [3]. In this section, we will give more detail for the transformation.

We will now illustrate the Tseytin transformation of the third constraint only for the columns, that is the first part of the expression:

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

Since the outer operator is already in conjunction, we only need to transform the inner part,

$$\left\{\neg\bigwedge_{i=1}^{2m}\left[(x_{ij_1}\wedge x_{ij_2})\vee(\neg x_{ij_1}\wedge\neg x_{ij_2})\right]\right\}.$$

We define the Tseytin variable a_i, b_i, c_i for $i = 1,\dots,2m$, 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 $i = 1,\dots,2m$, 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 $20m+1$ clause for the transformation of the inner part. Hence the CNF transformation of the third constraint will have $2\binom{2m}{2}(20m+1)$ clauses and each clause will have at most $2m+1$ literals.

2.4 Experiment and conclusion

From the CNF derivation of the constraints, we conclude that each constraint has a polynomial size in CNF as a function of the puzzle size. For the first constraint, we have $8m(2m-1)$ clauses and each clause has 3 literals.

Table 1: Comparison of the number of literals.

m	Polynomial formula	Exponential formula
2	4572	1824
3	17634	24768
4	52152	238912
5	117750	1894560
6	233268	13243680
7	419874	84840224
8	713328	509908608
9	1123830	2919287520

Table 2: Comparison of execution time (in seconds) for each method.

Size	Exp. formula		Exp. form. for 2^{nd} cons.		SMT		Polynomial formula	
	Pre-comp.	Solver	Pre-comp.	Solver	Pre-comp.	Solver	Pre-comp.	Solver
4×4	0.02	0.00	0.18	0.001	0.002	0.012	0.29	0.001
6×6	0.14	0.02	0.66	0.003	0.003	0.013	0.89	0.002
8×8	1.45	0.10	1.59	0.01	0.004	0.013	2.28	0.005
10×10	10.80	0.69	3.24	0.02	0.006	0.015	4.21	0.007
12×12	81.87	4.67	6.09	0.08	0.009	0.019	7.01	0.011
14×14	-	-	10.74	0.32	0.010	0.020	10.81	0.016
16×16	-	-	19.42	1.43	0.015	0.023	16.79	0.027
18×18	-	-	45.33	6.56	0.015	0.027	23.35	0.035

For the second constraint, the trivial exponential formula has $8m\binom{2m}{m+1}$ clauses where each clause has $m+1$ literals. Meanwhile, the recursive polynomial formula with the Tseytin transformation has at most $14 \cdot (2m)^2 \log_2(2m)$ clauses and each clause has $m+1$ literals. For small m, the trivial formula will have less literals than the recursive formula, but as soon as $m \geq 5$, the recursive formula has the advantage over the trivial formula in the number of literals.

For the third constraint, we also have two version in the CNF, the trivial exponential formula where it has $2 \cdot 2^{2m}\binom{2m}{2}$ clauses and each clause has $4m$ literals, and the polynomial CNF from the Tseytin transformation which has $2\binom{2m}{2}(20m+1)$ clauses where each clause has $4m$ literals. Here the polynomial CNF will have smaller number of literals if $m \geq 3$.

For the whole puzzle, the comparison of the number of the CNF literals is shown in Table 1. For brevity, we only compare up to $m = 9$.

We also did a simulation in solving a binary puzzle with various sizes. In this puzzle, around 70% of the cells were blank. We measure the computation time which is shown in Table 2. The dash in the table means that the computer is running out of memory.

For each case, we do 18 experiments, and then take the average timing. The result for the polynomial formula is shown in the last column of the Table 2. Here we only show the timing up to $m = 9$, but the largest puzzle we had solved so far using the polynomial formula is 34×34 with the pre-computation time equal to 159.65 second and the solver time is equal to 0.186 second. As a comparison, we show the result from our previous work in the second column [3] and the third column [1]. The result in the third column is acquired by applying satisfiability modulo theory (SMT) to solve the puzzle.

The experiment 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 7.5.1 as the wrapper and front-end programming language, CryptoMiniSat 2 for the SAT solver, and Yices 2.5 for the SMT solver.

References

[1] Putranto Utomo. Satisfiability modulo theory and binary puzzle. *Proc. The 2016 International Conference on Mathematics: Education, Theory & Application*, December 2016.

[2] Putranto Utomo and Ruud Pellikaan. On The Rate of Constrained Arrays. *Proc. IndoMS International Conference on Mathematics and Its Applications (IICMA) 2015*, 3 November 2015. `http://www.win.tue.nl/~ruudp/paper/75.pdf`

[3] Putranto Utomo and Rusydi Makarim. Solving the Binary Puzzle Proc. 22nd Conference on Application of Computer Algebra, August 2016.
Book of abstract: `http://www.mathematik.uni-kassel.de/ACA2016/docs/ACAproc.pdf`

[4] Putranto Utomo and Ruud Pellikaan, Binary Puzzles as an Erasure Decoding Problem. *Proc. 36th WIC Symp. on Information Theory in the Benelux*, pp. 129–134 May 2015. `http://www.w-i-c.org/proceedings/proceedings_SITB2015.pdf.`

[5] Marzio De Biasi, Binary puzzle is NP-complete, `http://www.nearly42.org/vdisk/cstheory/binaryp.pdf`

[6] Mate Soos. The CryptoMiniSat 5 set of solvers at SAT Competition 2016. In *Proceedings of SAT Competition 2016: Solver and Benchmark Descriptions - SAT Competition 2016*, page 28, 2016. Available at `https://github.com/msoos/cryptominisat`.

[7] Harald Seltner. Extracting Hardware Circuits from CNF Formulas. Master's thesis, Institute of Formal Models and Verification, Johannes Kepler Universität Linz, 2014. Available at `http://fmv.jku.at/master/Seltner-MastherThesis-2014.pdf`.

[8] Wikipedia. Conjunctive normal form — Wikipedia, The Free Encyclopedia Online; accessed 30-November-2016 `https://en.wikipedia.org/w/index.php?title=Conjunctive_normal_form&oldid=752290376`

[9] Grigorii Samuilovich Tseitin. On the Complexity of Derivation in Propositional Calculus, Automation of Reasoning 2: Classical Papers on Computational Logic. 1983, page 466–483, Springer Berlin Heidelberg, `http://dx.doi.org/10.1007/978-3-642-81955-1_28`