Satisfiability modulo theory and binary puzzle

To cite this article: Putranto Utomo 2017 J. Phys.: Conf. Ser. 855 012057

View the article online for updates and enhancements.

Related content

- Kinematic Labs with Mobile Devices: Working with spreadsheets
  J M Kinser

- Faster than classical quantum algorithm for dense formulas of exact satisfiability and occupation problems
  Salvatore Mandra, Gian Giacomo Guerreschi and Alán Aspuru-Guzik

- Cavity Dynamics behind a 2-D Wedge Analyzed by Incompressible and Compressible Flow Solvers
  Sunho Park, Heebum Lee, Hae-Kyu Choi et al.
Satisfiability modulo theory and binary puzzle

Putranto Utomo
Department of Mathematics and Computer Science, Eindhoven University of Technology; and Faculty of Mathematics and Natural Science, Sebelas Maret University
E-mail: p.h.utomo@tue.nl

Abstract. The binary puzzle is a sudoku-like puzzle with values in each cell taken from the set \{0, 1\}. We look at the mathematical theory behind it. A solved binary puzzle is an $n \times n$ binary array where $n$ is even that satisfies the following conditions:

1. No three consecutive ones and 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. Every two rows and every two columns must be distinct.

The binary puzzle had been proven to be an NP-complete problem [5].

Research concerning the satisfiability of formulas with respect to some background theory is called satisfiability modulo theory (SMT). An SMT solver is an extension of a satisfiability (SAT) solver. The notion of SMT can be used for solving various problem in mathematics and industries such as formula verification and operation research [1, 7].

In this paper we apply SMT to solve binary puzzles. In addition, we do an experiment in solving different sizes and different number of blanks. We also made comparison with two other approaches, namely by a SAT solver and exhaustive search.

1. Introduction
Recall the definition of binary puzzle in the abstract. Figure 1 is an example of initial setting of a binary puzzle. The unique solution satisfying all three conditions is given in Figure 2. Binary puzzles can be seen as constrained arrays [13]. One can also see this array from an erasure correcting point of view [12]. This paper focuses on solving binary puzzles.
We have solved the binary puzzle by three methods: exhaustive search, system of polynomials, and satisfiability (SAT) theory [14]. In this paper, we consider a new approach by means of satisfiability modulo theory (SMT). Using this approach, we improve a lot in terms of the size of the puzzle.

2. Satisfiability modulo theory (SMT)
One of the most fundamental problems in mathematics and computer science is the satisfiability problem, which deals with checking and finding a solution of a formula with some constraints [6]. Some of these problems can be expressed as a propositional satisfiability (SAT) problem which can be solved efficiently by a modern SAT solver. But, some other problems are too complex to be expressed in propositional SAT problem and more simple and easily expressed in first or higher order logic, which can include functions, and non-Boolean variables. The more complex a problem, the more difficult to be solved.

One way to deal with this problem is usually by focusing on fragments of first order logic that are restricted semantically, by constraining the interpretation of certain symbols to some logical background theory, such as the theory of equality, integer number, real number, bitvector, and so on. The satisfiability problem with these restrictions is called satisfiability modulo theory (SMT) [3].

2.1. SMT solver
Unlike SAT solvers, which are more mature and mostly have the same input and the same underlying theory, an SMT solver has more variance in terms of background theory behind the screen, depending on the problem it intends to solve. One of the reasons to make a restriction is to boost the performance of the solver. Nevertheless, there is an international initiative on SMT called SMT-LIB [2]. Two of its main purposes are to create standard descriptions of theories in SMT and to develop common input and output languages for SMT solvers.

In general, there are two approaches in designing an SMT solver, the lazy and the eager approach. The lazy approach is characterized by a combination of one or more theory solvers with a conflict-driven clause learning (CDCL) SAT engine, meanwhile the eager approach uses any technique so that the SMT problem is reduced to a propositional satisfiability (SAT) problem [3].

2.2. Related theories
We consider several specific theories that are often used in SMT.

The theory of equality with uninterpreted function (EUF), also known as the empty theory is the most general case of theory because there are no restrictions on the model. This problem can be solved in polynomial time using the congruence closure algorithm [1].

Figure 1: Unsolved Puzzle

Figure 2: Solved Puzzle
The next theory we would like to introduce is the arithmetic theory. The arithmetic theory is often divided into three categories, integer arithmetic, real arithmetic, and difference logic. The symbols used in the real and integer arithmetic are in \{+,-,\ast,\ge\}. The linear integer arithmetic problem, also known as Presburger arithmetic, can be solved using the Presburger algorithm. For real arithmetic, the simplex based algorithm is often used in practice for solving the satisfiability problem \[1\]. As stated in \[3\], difference logic expressions are in the form \(x - y \bowtie c\), where \(\bowtie\) is an element of \{=,\le,\ge\}, \(c\) is a constant, and \(x\) and \(y\) are variables.

Another theory we would like to mention is the theory of bitvector. The function and predicate symbol in this theory usually include extraction, concatenation, bitwise logical operators, and arithmetic operations \[1\].

For one who is interested in other theories such as arrays, strings, finite sets, we refer to \[3\] or any other survey on satisfiability modulo theory.

2.3. Combining theory
We often deal with a problem involving many theories at once. In solving this kind of problem, we need to combine the theory solver for the theories \(T_1, \ldots, T_n\) into one theory solver that solves the satisfiability modulo combination theory \(T_1 \oplus \cdots \oplus T_n\).

The idea for combination methods can be traced back to Nelson and Oppen. We refer to \[3\] for its procedure. One of the requirements using this kind of combining methods is that \(T_1, \ldots, T_n\) should be disjoint in the sense that all theories do not share function symbols.

One example of a framework for combining theories is the delayed theory combination (DTC). It is based on the conflict-driven clause learning (CDCL) SAT engine, which is also known as DPLL(\(T\))^1 algorithm, where \(T\) is a theory solver being used. The idea of DTC is to make an independent theory solver for each theory and extend the interface of the SAT engine so that it can interact with each theory solver using the Nelson-Oppen style \[3\].

Another way to combine one or more theories is by using Ackermann’s expansion. In this method, every function in the input formula is replaced by a fresh variable and their congruence constraints. The new formula will then be equisatisfiable and contain no uninterpreted function symbols \[1\].

We choose Yices 2 as the solver for the binary puzzle, since it focuses on satisfiability and model generator and that is exactly our problem we want to solve. The comparison and performance of Yices can be seen in \[10, 9, 8, 4\].

The design of the SMT solver in Yices 2 follows the DTC model. One can find the details in \[9\].

3. Solving a binary puzzle with SMT
Since the bitvector representation provides a more compact representation and often allows a more efficient solution \[1\], we use the bitvector representation for the problem of the binary puzzle. Suppose we want to solve an \(n \times n\) binary puzzle. First, we define \(2n\) bitvector variables for the columns and the rows, where each bitvector consists of \(n\) bits.

For the 1\textsuperscript{st} constraint, it is more natural to express every three consecutive cells as a number in their binary representation, for example, 011 is equal to 3 and 101 is equal to 5. Then we express the first constraint using bitvector arithmetic operator \less than\ and \greater than\ as explained in \[9\] as follows. Let \(x = x_1x_2x_3\) be any three consecutive cells, then \(x\) must be less than 111 = 7 and \(x\) must be greater than 000 = 0.

For the 2\textsuperscript{nd}, balancedness constraint, we use integer arithmetic, since we need to count the number of ones in each row and each column to check whether the number of ones is equal to the number of zeros.

---

1 DPLL is an abbreviation of Davis, Putnam, Logemann, and Loveland
For the 3rd constraint, we use the operator distinct defined in [9] for all the columns and for all the rows so that there is no identical pair of columns and also no identical pair of rows.

Finally, we also need to impose the additional constraint to the row and column variables, so that they are consistent in the sense that the \( i^{th} \) bit on column \( j \) should be equal to the \( j^{th} \) bit on row \( i \).

4. Experiment

In this section, we will briefly discuss the simulation of solving the binary puzzle. 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 we used for the experiment are Sagemath 7.2 as the wrapper and front-end programming language; and Yices 2.5 as the SMT solver.

We prepared a database of solved binary puzzle with size ranging from 4 \( \times \) 4 until 100 \( \times \) 100. For each size, we have 6 distinct solved puzzles. In this research, we also interested whether there is a differences in the computation time for different puzzles. In our experiment, we hardly see any difference in the computation time, as we can see in Figure 3. In this figure, different colors represent different puzzles.

The procedure for acquiring a partially solved binary puzzle is as follow. Suppose we want an \( n \times n \) puzzle with \( k \) blanks, where \( 0 < k \leq n^2 \) and \( n = 2m \). First, we pick a particular puzzle from the database and then choose \( k \) random cells from the \( n \times n \) arrays to be erased.

We compare the execution time with the two other approaches for solving the binary puzzle, namely exhaustive search and propositional SAT solver. One can refer to [14] for the details of this approach.

In this research, we also did an experiment regarding the distribution of uniqueness. For each number of blanks, we calculate the ratio of uniqueness, that is the number of experiment having a unique solution divided by the total experiment. For 40 \( \times \) 40, we refer to Figure 6. If the ratio is above 0.5, then the puzzle is more likely to have a unique solution. Table 1 shows when the ratio hits 0.5 for different sizes.

5. Conclusion

The result from this experiment is quite surprising, since it can solve binary puzzles beyond our expectation.

We have several remarks:

(i) The limit of the SAT solver with the current hardware specification is 14 \( \times \) 14 since the precomputation blows up exponentially.

(ii) The computation time using the SMT solver is an exponential function \( y = e^{-3.5597+0.0028x} \) in terms of the number of blanks. See Figure 5.

(iii) The comparison of different solvers and different sizes with 70% of blanks is given in Table 2. In this case, exhaustive search will fail to find a solution for 16 \( \times \) 16 puzzle in a reasonable time compared to SMT solver.

(iv) The exhaustive search methods can catch up with the SMT solver for very small portion of blanks, but as the blanks get more, the exhaustive search cannot find a solution in a reasonable time compared to the SMT solver. For the case of 10% of blanks, the limit for exhaustive search is 76 \( \times \) 76, and comparison can be seen in Figure 4.

(v) Since the number of blanks does not significantly affect the size of the solver input, it is obvious that the precomputation time is not affected by the increase of the blanks, as we can see in Figure 7.
Table 1: Number of blanks where the ratio of uniqueness is equal to 0.5.

<table>
<thead>
<tr>
<th>Size</th>
<th>Number of blanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 × 4</td>
<td>9</td>
</tr>
<tr>
<td>6 × 6</td>
<td>19</td>
</tr>
<tr>
<td>8 × 8</td>
<td>30</td>
</tr>
<tr>
<td>10 × 10</td>
<td>44</td>
</tr>
<tr>
<td>12 × 12</td>
<td>53</td>
</tr>
<tr>
<td>14 × 14</td>
<td>67</td>
</tr>
<tr>
<td>16 × 16</td>
<td>80</td>
</tr>
<tr>
<td>18 × 18</td>
<td>92</td>
</tr>
<tr>
<td>20 × 20</td>
<td>110</td>
</tr>
<tr>
<td>22 × 22</td>
<td>126</td>
</tr>
<tr>
<td>24 × 24</td>
<td>133</td>
</tr>
<tr>
<td>26 × 26</td>
<td>150</td>
</tr>
<tr>
<td>28 × 28</td>
<td>164</td>
</tr>
<tr>
<td>30 × 30</td>
<td>188</td>
</tr>
<tr>
<td>32 × 32</td>
<td>209</td>
</tr>
<tr>
<td>34 × 34</td>
<td>238</td>
</tr>
<tr>
<td>36 × 36</td>
<td>238</td>
</tr>
<tr>
<td>38 × 38</td>
<td>248</td>
</tr>
<tr>
<td>40 × 40</td>
<td>279</td>
</tr>
</tbody>
</table>

Figure 3: Comparisson time between different puzzle for 40 × 40 puzzle.
Figure 4: Comparison time between SMT solver and exhaustive search.

<table>
<thead>
<tr>
<th>Size</th>
<th>SAT (sec)</th>
<th>EXH (sec)</th>
<th>SMT (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 × 4</td>
<td>0.007</td>
<td>0.022</td>
<td>0.015</td>
</tr>
<tr>
<td>6 × 6</td>
<td>0.075</td>
<td>0.179</td>
<td>0.017</td>
</tr>
<tr>
<td>8 × 8</td>
<td>0.690</td>
<td>0.149</td>
<td>0.018</td>
</tr>
<tr>
<td>10 × 10</td>
<td>5.615</td>
<td>0.526</td>
<td>0.021</td>
</tr>
<tr>
<td>12 × 12</td>
<td>39.464</td>
<td>53.986</td>
<td>0.028</td>
</tr>
<tr>
<td>14 × 14</td>
<td>244.815</td>
<td>129.265</td>
<td>0.031</td>
</tr>
<tr>
<td>...</td>
<td>-</td>
<td>-</td>
<td>:</td>
</tr>
<tr>
<td>36 × 36</td>
<td>-</td>
<td>-</td>
<td>0.268</td>
</tr>
<tr>
<td>38 × 38</td>
<td>-</td>
<td>-</td>
<td>0.396</td>
</tr>
<tr>
<td>40 × 40</td>
<td>-</td>
<td>-</td>
<td>0.448</td>
</tr>
</tbody>
</table>

Table 2: Comparison of different solver and different sizes

Figure 5: Number of blanks vs. computation time for SMT solver for 40 × 40 puzzle (log. scale)
Figure 6: Number of blanks vs. ratio of uniqueness

Figure 7: Number of blanks vs. precomputation time for SMT solver
References


