Exploring Properties of the Instant Insanity Puzzle with Constraint Satisfaction Approach

Sven Löffler, Ke Liu[a] and Petra Hofstedt

Department of Mathematics and Computer Science, MINT, Programming Languages and Compiler Construction Group, Brandenburg University of Technology Cottbus-Senftenberg, Konrad-Wachsmann-Allee 5, 03044 Cottbus, Germany

Keywords: Constraint Programming, CSP, CSOP, Optimization, Instant Insanity Puzzle, Problem Generation, Test Case Generation.

Abstract: The Instant Insanity Puzzle is a challenging and interesting puzzle of combinatorial nature. The puzzle consists of four different cubes where each face of each cube has one of the four colors red, green, white and blue. The goal is to arrange the cubes in a tower with dimensions $1 \times 1 \times 4$ such that on each of the four long sides of the tower, every color appears exactly once. In this paper we pose questions derived from the puzzle, but with increased difficulty and generality. Amongst other things, we try to find a new problem instance (a new color assignment for the cubes) such that the number of solutions of the instant insanity puzzle is minimized but not null. In addition, we also present a constraint programming model for the proposed questions, which can provide the answers to our questions. The purpose of this paper is on the one hand to share our results over the instant insanity puzzle, and on the other hand to share our gained knowledge on finding problems by constraining the solutions of constraint satisfaction problems, which is (amongst other things) useful for the generation of test data and teaching material.

1 INTRODUCTION

The puzzle game Instant Insanity was popularized by the Parker Brothers Company in the late 1960s. It has a rich history — the name Instant Insanity dates back to the 1960s, but there were many earlier variants, released under such names as Katzenjammer, Groceries, and The Great Tantalizer.

The Instant Insanity puzzle consists of four different cubes where each face of each cube has one of the four colors red, green, white and blue. The goal is to arrange the cubes in a tower with dimensions $1 \times 1 \times 4$ such that on each of the four long sides of the tower, every color appears exactly once. Figure 1 shows a modern version of the Instant Insanity puzzle.

The cube nets with the color assignments for each face of the original Instant Insanity game is shown in Figure 2. Each cube has 24 possible piece configurations: 6 ways to choose the side that faces down, and 4 possible ways to rotate the visible faces.

We believe that the Instant Insanity puzzle is a proper research object for the constraint programming community because of its combinatorial and symmetrical nature. Thus, we defined the following interesting questions about the Instant Insanity puzzle which pose attractive challenges:

Figure 1: A modern version of the Instant Insanity puzzle.

1. Can we find a suitable CSP, which solves the original problem using constraint programming? In this paper, we call all solutions of the original problem an `all different Instant Insanity solution` (ADIIS).
2. Do the cubes of the original problem allow to find a tower where each of the long sides of the tower use the same color, such that one long side is green, one red, one blue and one white? In this

[a] https://orcid.org/0000-0002-5256-9253

Löffler, S., Liu, K. and Hofstedt, P.
Exploring Properties of the Instant Insanity Puzzle with Constraint Satisfaction Approach.
DOI: 10.5220/0008880403310338
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 331-338
ISBN: 978-989-758-395-7; ISSN: 2184-433X

paper, we call each solution of this kind an `all equal Instant Insanity solution` (AEIIS). If there is no such solution, can we then find a set of cubes which has solutions of both kinds: ADIIS and AEIIS?

3. Can we find a set of cubes, which has the lowest number of ADIIS but at least one?
4. Can we find a set of cubes, which has the lowest number of AEIIS but at least one?
5. Can we find a set of cubes, which has the lowest number of ADIIS plus AEIIS but at least one of ADIIS and one of AEIIS?

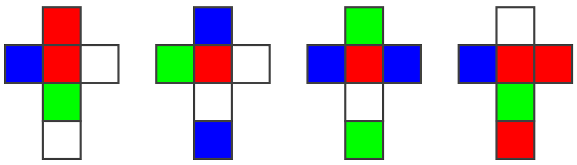


Figure 2: The cube nets with collor assignments of the Instant Insanity puzzle.

Remark 1: The questions three, four and five are motivated by finding a new puzzle, which is as difficult as possible. We assume that such a problem is more complicated if the ratio of solutions by possible combinations is as small as possible.
Remark 2: For the questions two to five, we have the additional constraint, that each cube should contain at least each color once.
Remark 3: The questions two to five assume that we find a possibility to define constraints over the solutions of a CSP. We think that such an approach can be useful in different areas like the development of games or logic puzzles, in test case generation or in the generation of educational material.

The rest of this paper is organized as follows. In Section 2, we give a brief introduction into constraint programming. Afterwards, in Section 3, we describe our constraint models to solve the posed questions and present the experimental results. In Section 4 we conclude and give a brief overview on future works.

2 PRELIMINARIES

In this section we give some basic definitions and concepts of constraint programming (CP) and the relevant constraints for the modelling of the Instant Insanity puzzle.

Constraint programming is a powerful technique to tackle (generally NP-hard) combinatorial problems. Since the 2000s, constraint programming is also used for test case generation (Aichernig and Salas, 2005; Degrave et al., 2009; Caballero et al., 2010; Wotawa et al., 2011; Ernsting et al., 2012). We like to use constraint programming in a similar way for creating new tasks during we define constraints over the solutions of the searched tasks.

We consider constraint satisfaction problems (CSPs), which are defined in the following way.

CSP (Dechter, 2003) A constraint satisfaction problem (CSP) is defined as a 3-tuple $P = (X, D, C)$ with $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables, $D = \{D_1, D_2, \ldots, D_n\}$ is a set of finite domains where D_i is the domain of x_i and $C = \{c_1, c_2, \ldots, c_m\}$ is a set of constraints covering between one and all variables in X.

Each constraint c_i is a relation defined over a subset of the variables X and restricts the values that can be simultaneously assigned to these variables. A solution of a CSP P is a complete instantiation satisfying all constraints of the CSP P. Further we define constraints required to describe the Instant Insanity puzzle. Let a CSP $P = (X, D, C)$ and a subset X' of variables X of the CSP P be given.

The **arithm** constraint describes an arithmetic relation between two ($x_i, x_j \in X'$) or three ($x_i, x_j, x_k \in X'$) variables (see Equation 1).

$$\begin{aligned} arithm(x_i, \Re, x_j) &:= x_i \ \Re \ x_j \\ arithm(x_i, \odot, x_j, \Re, x_k) &:= x_i \odot x_j \ \Re \ x_k \end{aligned} \tag{1}$$

The symbol $\Re$ is an operator in $\{=, <, >, \neq, \leq, \geq\}$ and the symbol $\odot$ is an operator in $\{+, -, *, /\}$.

The **allDifferent** constraint is a prime example for a global constraint. For an ordered set of variables $\{x_1, x_2, \ldots, x_n\} = X'$ it guarantees that each variable has a different value (see Equation 2).

$$allDifferent(X') := x_i \neq x_j \ \forall x_i, x_j \in X', i \neq j \tag{2}$$

It outlines two of the main advantages of global constraints. On one hand, using the $allDifferent$ constraint simplifies the modelling process only one $allDifferent$ constraint must be posted instead of many pairwise not equal constraints and on the other hand the $allDifferent$ constraint allows to use more effective propagation algorithms (López-Ortiz et al., 2003) as the pairwise not equal constraints. Analogously to the $allDifferent$ constraint there is an $allEqual$ constraint, which requires that every variable of a sequence of variables X' must have the same value.

The **count**[2] constraint is a common global constraint, where for an ordered set of variables $X' = \{x_1,$

[2]This paper follows the naming convention of Choco solver. Other solvers might use a different name for the same constraint.

$x_2, \ldots, x_n\}$ the variable $occ \in X$ with domain $D_{occ} = \{occ_{min}, \ldots, occ_{max}\}$ denotes the admissible number of occurrences of the value $v \in \mathbb{N}$ in X' (see Equation 3).

$$count(X', occ, v) := (\sum_{x \in X'} \begin{cases} 0 & x \neq v \\ 1 & x = v \end{cases}) \in D_{occ} \quad (3)$$

The **element** constraint guarantes for a sequence of variables $\{x_1, x_2, \ldots, x_n\} = X'$, that the ith variable must be equal to a variable $x' \in X$ (see Equation 4).

$$element(x', X', i) := X'[i] == x' \quad (4)$$

The **table** constraint is also one of the most frequently used constraints in practice. For an ordered subset of variables $X' = \{x_i, \ldots, x_j\}$, a positive (or negative) *table* constraint defines that any solution of the CSP P must (not) be explicitly assigned to a tuple of a given tuple list T, which consists of the allowed (disallowed) combinations of values for X' (see Equation 5).

$$\begin{aligned} table(X', T) := \{(x_i, \ldots, x_j) \mid \\ x_i \in D_{(x_i)}, \ldots, x_j \in D_{(x_j)}\} \in T \end{aligned} \quad (5)$$

The **sum** constraint is also a common global constraint, where the sum of a sequence of variables $(x_i, \ldots, x_j)$ must be in relation $\Re \in \{=, <, >, \neq, \leq, \geq\}$ to a variable $s \in X$ (see Equation 6).

$$sum(x_i, \ldots, x_j, \Re, s) := x_i + \ldots + x_j \; \Re \; s \quad (6)$$

The **regular** constraint (Hellsten et al., 2004; Pesant, 2001; Pesant, 2004) is the last global constraint we present in this short selection. For the *regular* constraint, we briefly need to show the definition of a deterministic finite automaton (DFA) (Hopcroft and Ullman, 1979).

A deterministic finite automaton (DFA) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the finite input alphabet, δ is a transformation function $Q \times \Sigma \rightarrow Q$, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final or accepting states. A word $w \in \Sigma^*$ is accepted by M, i.e. $w \in L(M)$, if the corresponding DFA M with the input w stops in a final state $f \in F$ (Hopcroft and Ullman, 1979).

The *regular* constraint obtains a DFA $M = (Q, \Sigma, \delta, q_0, F)$ and an ordered subset of variables $X' = \{x_1, \ldots, x_n\} \subseteq X$, with $D(x_i) \subseteq \Sigma$ for $1 \leq i \leq n$ as input and guarantees that every sequence $d_1 \circ \ldots \circ d_n$ of values for $x_1, \ldots, x_n$ must be a word of the regular language recognized by the DFA M, where $\circ$ is the concatenation of two words (Pesant, 2004) (see Equation 7).

$$\begin{aligned} regular(X', M) = \{(d_1, \ldots, d_n) | \forall i \; d_i \in D_i, \\ d_1 \circ d_2 \circ \ldots \circ d_n \in L(M)\} \end{aligned} \quad (7)$$

We refer to (Dechter, 2003; Rossi et al., 2006; Lecoutre, 2009) for more comprehensive and profound introduction to constraint programming.

3 THE CONSTRAINT PROGRAMMING MODELS

In this section we explain the constraint models we used to answer the questions 1 to 5 and present our results.

3.1 A Model for the ADIIS and AEIIS Problem

To solve problem 1, we first should identify the decision variables for the CSP model. Then we impose constraints on these variables based on the problem definition. Focusing on the Instant Insanity puzzle it has 4 cubes with 6 faces each, we use a two-dimensional array of integer variables $X = \{\{x_{1,1}, \ldots, x_{1,6}\}, \ldots, \{x_{4,1}, \ldots, x_{4,6}\}\}$ to represent the colors of each side of each cube, each of which has domain $d_{i,j} = \{0, 1, 2, 3\} | \forall i \in \{1, \ldots, 4\}, j \in \{1, \ldots, 6\}$ representing the four different colors.

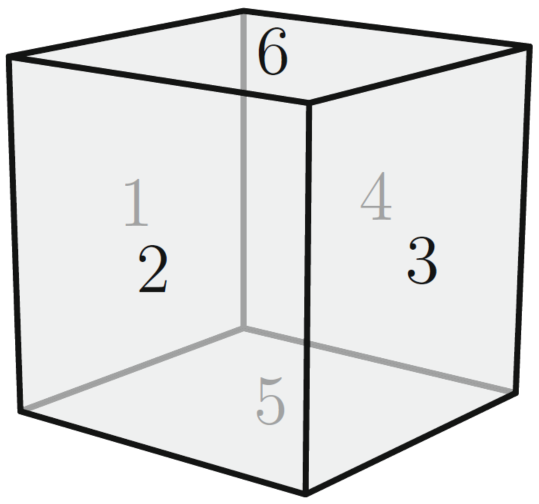
Suppose that we number the sides as depicted in Fig. 3. Then there are 24 possible piece configurations: 6 ways to choose the side that faces down, and 4 possible ways to rotate the visible faces. These 24 piece configurations are also listed in Fig. 3.

Therefore, each sequence of variables which describes one cube ($\{x_{i,1}, \ldots, x_{i,6}\} \forall i \in \{1, \ldots, 4\}$) must satisfy one of these 24 possible piece configurations. It is not possible to use this configurations directly, but we can substitute the values 1 to 6 of the configurations with the numbers 1 to 4, which represent the colors (1 = RED, 2 = BLUE, 3 = GREEN, 4 = WHITE) of the cubes [3]. Using the numeration of Fig. 3 and the color net from Fig. 2, the puzzle has the following color sequences $(C_1, \ldots, C_4)$ for the 4 cubes:

1. $RED, RED, GREEN, WHITE, BLUE, WHITE$
2. $BLUE, RED, WHITE, BLUE, GREEN, WHITE$
3. $GREEN, RED, WHITE, GREEN, BLUE, BLUE$
4. $WHITE, RED, GREEN, RED, BLUE, RED.$

Let be T the 24×6 matrix which represents the 24 6-tuples in Fig. 3. Therefore, the *table* constraint with matrix T' and variables $\{x_{i,1}, \ldots, x_{i,6}\} \forall i \in \{1, \ldots, 4\}$ guarantees that each face of the cube i has a color,

[3] In the rest of the paper, we use the numerical values 1, 2, 3, and 4 to represent the corresponding colors RED, BLUE, GREEN and WHITE.



(1,2,3,4,5,6), (1,5,3,6,4,2), (1,4,3,2,6,5), (1,6,3,5,2,4), (2,3,4,1,5,6), (2,5,4,6,1,3), (2,1,4,3,6,5), (2,6,4,5,3,1), (3,2,1,4,6,5), (3,5,1,6,2,4), (3,4,1,2,5,6), (3,6,1,5,4,2), (4,3,2,1,6,5), (4,6,2,5,1,3), (4,1,2,3,5,6), (4,5,2,6,3,1), (5,2,6,4,3,1), (5,3,6,1,4,2), (5,4,6,2,1,3), (5,1,6,3,2,4), (6,2,5,4,1,3), (6,3,5,1,2,4), (6,4,5,2,3,1), (6,1,5,3,4,2)

Figure 3: Labels for each of the faces on a cube.

which is reachable by rotating the cube i along the x-, y-, or z-axis (see Eq. 8).

$$table(\{x_{i,1},\ldots,x_{i,6}\},T^i)\,\forall i\in\{1,\ldots,4\} \tag{8}$$

Where table T^i is defined by the possible piece configurations T and the given colors $C_1,\ldots C_4$ of the Instant Insanity puzzle (see Eq. 9).

$$T^i_{x,y}=C_i[T_{x,y}]$$
$$\forall i\in\{1,\ldots,4\},x\in\{1,\ldots,24\},y\in\{1,\ldots,6\} \tag{9}$$

Remark 4: Using the table constraint like presented before, leads to a strict cube order C_1,C_2,C_3,C_4, which is not given in the game. The patterns we try to find (ADIIS or AEIIS) are independent from the cube order. Thus, a solution for the given variable order is always also a solution for every other permutation of the 4 cubes. Doing this reduces the number of solutions and the search space by factor $\frac{1}{4!}=\frac{1}{24}$, which leads to a speed up in the solution process. If all solutions are searched, then the real solution number is the number of detected solutions times 24. We call such a constraint a symmetry breaking constraint.

After constraining the color side dependencies of the cubes, it is necessary to define the target requirement. For ADIIS, every side of a tower must use every color once. Thus all equally positioned sides of the different cubes must take distinct colors. Therefore, $(x_{1,i},x_{2,i},x_{3,i},x_{4,i}\,\forall i\in\{1,2,3,4\})$ must satisfy the allDifferent constraint, given by Eq. 10:

$$allDifferent(x_{1,i},x_{2,i},x_{3,i},x_{4,i})\,\forall i\in\{1,2,3,4\} \tag{10}$$

For the AEIIS problem we need instead of the constraint given by Eq. 10 an *allEqual* constraint (see Eq. 11) for all variables $x_{1,i},x_{2,i},x_{3,i},x_{4,i}\,\forall i\in\{1,2,3,4\}$.

$$allEqual(x_{1,i},x_{2,i},x_{3,i},x_{4,i})\,\forall i\in\{1,2,3,4\} \tag{11}$$

Results

After presenting our CSP model for finding an ADIIS and an AEIIS, we want to show our results.

All the experiments (for all 5 problem instances) are set up on a DELL laptop with an Intel i7-4610M CPU, 3.00GHz, with 16 GB RAM, 1600 MHz DDR3 and running under Windows 7 professional with service pack 1. The algorithms are implemented in Java under JDK version 1.8.0_191 and Choco Solver (Prud'homme et al., 2016). We used the *DowOverWDeg* search strategy which is explained in (Boussemart et al., 2004) and is used as the default search strategy in the Choco Solver (Prud'homme et al., 2016).

For the ADIIS problem instance, we could find all solutions in less than one second. There are eight solutions in which every four solutions are equivalent under rotation of the tower around the z-axis. The two real different solutions (with respect to rotation) are shown in Figure 4.

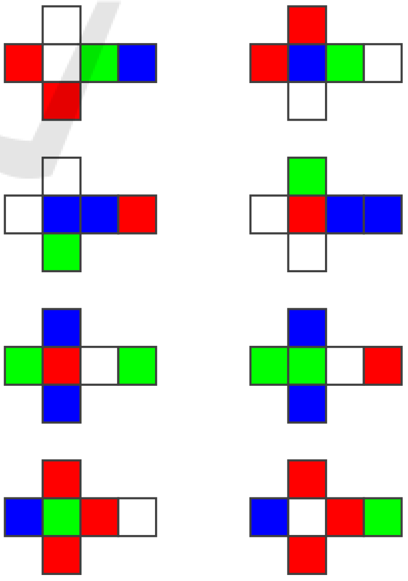


Figure 4: The two different solutions of the Instant Insanity puzzle.

For the AEIIS problem instance the Choco Solver can detect that no solution exists. This leads us to the second part of question two; can we find a set of cubes which has solutions of both ADIIS and AEIIS?

3.2 A Model for Finding Cubes with ADIIS and AEIIS

For finding cubes that have at least one ADIIS and one AEIIS, we have to remodel our CSP significantly. Finally, again, we only need for each cube, for each side a color assignment: RED, BLUE, GREEN or WHITE. But for this, we have to respect that a rotation of the cubes should have an ADIIS and at least one other rotation an AEIIS. So we formulate constraints over the solutions of our original problem.

Again, we use a two-dimensional array of integer variables $X = \{\{x_{1,1},...,x_{1,6}\},...,\{x_{4,1},...,x_{4,6}\}\}$ to represent the colors of each side of each cube, each of which has domain $d_{i,j} = \{0,1,2,3\} | \forall i \in \{1,...,4\}, j \in \{1,...,6\}$ representing the four different colors. We additionally post constraints that guarantee, that each cube has each color at least one time (see Equation 12).

$$\begin{gathered} count(\{x_{i,1},...,x_{i,6}\},\{1,2,3\},v) \\ \forall i \in \{1,...,4\}, v \in \{1,...,4\} \end{gathered} \tag{12}$$

The difficulty is that this time we have no input tables T^i for our *table* constraints in Eq. 8. We only know the structure of the tables T^i but not the concrete content. We know that T^i must satisfy the rotations shown in Figure 3, which gives us the following information:

1. The table T^i is a 24×6 matrix.
2. All 1s must be substituted by the same single color RED, BLUE, GREEN or WHITE. The same is valid for the other numbers 2 to 6.
3. If the color assignment for the first row of the matrix is clarified, then all other rows can be set instantly as consequence of this, because each number 1 to 6 is assigned to exactly one color (RED, BLUE, GREEN or WHITE).

Because we don't know the concrete values of the matrices T^i, but we know, that each T^i is a 24×6 matrix with values 1 to 4. We represent them as two-dimensional arrays of integer variables, where the first row of each matrix is the sequence of variables representing the ith cube $T^i_1 = \{x_{i,1},...,x_{i,6}\}$. Each other entry in T^i is a copy of one of the variables in the first line of T^i, where $T^i_{x,y} = x_{i,T_{x,y}}$ $\forall i \in \{1,...,4\}$, $x \in \{2,...,24\}$, $y \in \{1,...,6\}$, where T is the possible piece configuration given in Figure 3.

Next we have to formulate one solution of the newly formulated tables T^i. For this we create 16 new variables $S^{ADIIS} = \{s^{ADIIS}_{x,y} | \forall x \in \{1,...,4\}, y \in \{1,...,4\}\}$, where x represents the cube and y the side of the cube. The sides 5 (bottom side) and 6 (top side) of each cube are not important for the solution of the problem so that we do not consider them here.

For each cube i must exist one integer variable *index* with $D_{index} = \{1,...24\}$ such that all values of the variables $s^{ADIIS}_{i,*}$ are equal to the values of $T^i_{index,*}$. The *element* constraints shown in Equation 13 realize this.

$$element(s^{ADIIS}_{i,y}, T^i_{*,y}, index)\ \forall y \in \{1,...,4\} \tag{13}$$

The variables S^{ADIIS} represent a valid cube configuration (because of the *element* constraints), but not necessarily an ADIIS of the problem. To get an ADIIS it is necessary to post the $allDifferent$ constraints shown in Equation 10 on the variables $s^{ADIIS}_{1,i}, s^{ADIIS}_{2,i}, s^{ADIIS}_{3,i}, s^{ADIIS}_{4,i}$ instead of the variables $x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i}$.

After we guarantee that a solution for ADIIS exists, we have to do the same for AEIIS. So we create 16 new integer variables $S^{AEIIS} = \{s^{AEIIS}_{x,y} | \forall x \in \{1,...,4\}, y \in \{1,...,4\}\}$ and an *index* variable for each cube i with domain $D_{index} = \{1,...24\}$ such that all values of the variables $s^{AEIIS}_{i,*}$ are equal to the values of $T^i_{index,*}$. It is only necessary to change the S^{ADIIS} variables with the S^{AEIIS} variables in the *element* constraints shown in Equation 13 to realize this.

Analogously to ADIIS, it is only necessary to post the *allEqual* constraints shown in Equation 11 on the variables $s^{AEIIS}_{1,i}, s^{AEIIS}_{2,i}, s^{AEIIS}_{3,i}, s^{AEIIS}_{4,i}$ instead of the variables $x_{1,i}, x_{2,i}, x_{3,i}, x_{4,i}$ to get an AEIIS.

Results

Solving the newly created CSP delivers us a huge number of cube combinations which have an ADIIS and an AEIIS. However because the Instant Insanity puzzle is interesting because of its difficulty, we also want an cube configuration which has an ADIIS and an AEIIS but is also difficult to solve. We expect that such a puzzle is hard to solve if it has only a small number of solutions. This leads us to our questions 3, 4 and 5. Can we find a cube configuration such that it has a minimum number of ADIIS, a minimum number of AEIIS or a minimum number of ADIIS plus AEIIS?

3.3 A Model for Finding Cubes with a Minimum Number of ADIIS and AEIIS

For counting and minimizing the number of solutions we create two automatons, which represent all possible ADIIS and AEIIS without respect to the cube rotations. Then we create meta constraints, which represent these automatons and count the number of accepting paths inside them considering the cube rotations. The *index* variables and the S^{AEIIS} respectively the S^{ADIIS} variables, and the *element* constraints, shown in Equation 13 are no longer needed.

The automaton for the AEIIS over the variables $X^{AEIIS} = \{x_{1,1}, x_{2,1}, x_{3,1}, x_{4,1}, x_{1,2}, ..., x_{4,4}\}$ is created by clever intersections of singleton automatons M^{eq} which accept the words $\{0000, 1111, 2222, 3333\}$. For this we create four M_i^{eq} automatons over the variables $\{x_{i,1}, x_{i,2}, x_{i,3}, x_{i,4}\}$ $\forall i \in \{1,2,3,4\}$, fill them with the missing variables of the variable sequence X^{AEIIS} in a way that the missing variables can have any value of their domain, and intersect these four M_i^{eq} automatons to one M^{AEIIS} automaton. The automaton M^{ADIIS} for the ADIIS can be created analogously.

For the next step we need count variables $X^c = \{x_{i,a,b,c,d}\ \forall i,a,b,c,d \in \{1,2,3,4\}\}$ for each cube i and each combination of colors for the four visible sides $\{a,b,c,d\}$ with domains $\{0,1,2,3,4\}$. The variables represent how often each sequence of colors $\{a,b,c,d\}$ is included in the T^i rotation matrix of each cube i. Because of the structure of the rotation matrices T^i, each color sequence can occur between 0 and 4 times. We created a regular constraint for each cube i which counts the equal sequences of colors $\{a,b,c,d\}$ and sets the variables $x_{i,a,b,c,d}$ to the corresponding values.

In the next step we reduce all paths in M^{ADIIS} (respectively M^{AEIIS}) of the form $q \xrightarrow{a} q' \xrightarrow{b} q'' \xrightarrow{c} q''' \xrightarrow{d} q''''$ to the path $q \xrightarrow{abcd} q''''$, where $abcd$ is the concatenation of elements of the domain values of the variable sequence $\{x_{1,1}, ..., x_{4,1}\}$. We repeat this reduction for the paths over the variables $\{x_{1,i}, ..., x_{4,i}\}$ $\forall i \in \{2,3,4\}$. The result is an equivalent automaton M^{ADIIS}_{short} (respectively M^{AEIIS}_{short}) which represents the same words, with the difference that the inputs are not singleton colors {RED, BLUE, GREEN, WHITE} but instead color sequences of size four.

Figure 5 shows an example for clarifying the reduction process. For simplification only each two inputs are reduced to one.

Now we create meta constraints which combine the automatons M^{ADIIS}_{short} (respectively M^{AEIIS}_{short}) with the

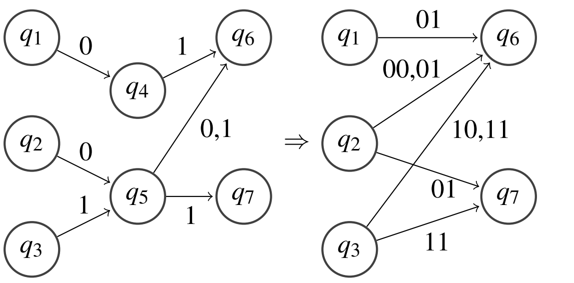


Figure 5: An example for the reducing process of an automaton.

count variables X^c. For each cube i, each value $abcd$ in the automaton M^{ADIIS}_{short} (respectively M^{AEIIS}_{short}) corresponds to the variable $x_{i,a,b,c,d}$, where the value of $x_{i,a,b,c,d}$ indicates how many cube rotations in T^i have the value sequence $abcd$.

Thus, each path in M^{ADIIS}_{short} (respectively M^{AEIIS}_{short}), from the start node to the end node with values $a_1b_1c_1d_1, a_2b_2c_2d_2, a_3b_3c_3d_3, a_4b_4c_4d_4$, where the value of all corresponding count variables $(x_{1,a_1,b_1,c_1,d_1}, x_{2,a_2,b_2,c_2,d_2}, x_{3,a_3,b_3,c_3,d_3}, x_{4,a_4,b_4,c_4,d_4})$ is not null, is exactly an ADIIS (respectively AEIIS) of the Instant Insanity problem with four cubes, where the six sides have the color assignment $\{x_{i,1}, ..., x_{i,6}\ \forall i \in \{1,2,3,4\}\}$. Furthermore, the result of the multiplication of the corresponding count variables $(x_{1,a_1,b_1,c_1,d_1} * x_{2,a_2,b_2,c_2,d_2} * x_{3,a_3,b_3,c_3,d_3} * x_{4,a_4,b_4,c_4,d_4})$ is exactly the number of possible cube rotations which reach this solution.

For the enumeration of all solutions, we create several constraints shown in Equation 14. For each state q_n of the automaton M^{ADIIS}_{short} a variable x_n is created, which counts the paths from the initial state of the automaton to q_n. The number of possibilities is calculated by the sum of the value of each predecessor x_{pre} multiplied by the number of cube rotations $x_{i,a,b,c,d}$, which represents the path from the predecessor node x_{pre} to the current node q_n with the value $abcd$. The variable $x_{initial}$, which represents the initial state $q_{initial}$ is set to 1. All other values are then calculated by the given constraints. The number of possible solutions is then exactly the value of the variable x_{final} which represents the singleton final state q_{final}.

$$\forall\ States\ q_n \in \{M^{ADIIS}_{short}\}:$$
$$x_n = \sum\{x_{pre} * x_{i,a,b,c,d} \mid \forall a,b,c,d, \quad \text{where } q_{pre} \xrightarrow{abcd} q_n\} \tag{14}$$

For example, if the color assignments *RED*, *RED*, *BLUE*, *GREEN*, *WHITE*, *BLUE* and *RED*, *RED*, *BLUE*, *GREEN*, *BLUE*, *WHITE* are in T^1, then the count variable $x_{1,1,2,3,4}$ has value 2 and there exists a constraint $q_{initial} \xrightarrow{1234} q_n$ with $x_{initial} = 1$, $x_n = x_{initial} * x_{1,1,2,3,4} = 1 * 2 = 2$, which set x_n to 2.

For minimizing the number of ADIIS, we have to minimize the x_{final} variable. For minimizing the number of AEIIS, we have to minimize the x_{final} variable, which results from constraints shown in Equation 14, where the M^{ADIIS}_{short} automaton is substituted by the M^{AEIIS}_{short} automaton. For finding the minimum number of ADIIS plus AEIIS, we have to create the constraints shown in Equation 14 for both automatons (M^{ADIIS}_{short} and M^{AEIIS}_{short}) and minimize the sum of both final states. In all cases we allow only values greater than null for the final states.

Result

The good news is that we were able to find solutions for all three instances. We found in less than 5 seconds a cube coloring which has only four ADIIS. In less than 3 seconds we found a cube coloring, which has only two AEIIS. In less than 4 seconds we found a cube coloring, which has only 24 ADIIS and two AEIIS. The given results are the number of solutions after removing equivalent solutions, which result from the rotation of the whole tower around the z-axis.

The bad news is that we unfortunately could not show that these are the minimum numbers of solutions after running the programs for two days. Nevertheless considering the found results, we can see that these are at least very good solutions, and maybe the best in case of AEIIS, and ADIIS plus AEIIS.

4 CONCLUSION AND FUTURE WORK

By means of the constraint programming approach, we found solutions for the original Instant Insanity problem and have proved that no AEIIS exists. However, we were not able to answer the rest of the questions posed in the Introduction (cf. Sec. 1), although some feasible solutions were found. We expect that we can also find the minimized solutions for our questions 3, 4 and 5 with our given CSP after including some optimizations like using parallel computing and minimizing the number of used variables and constraints, especially the ones used in Equation 14.

The real benefits of this paper are the abstractions of the *table* and the *regular* constraints, to find a problem of a defined kind with special requirements for the solutions. Examples for these special requirements are the demand to have solutions of one or many special kinds (in this case ADIIS and AEIIS) or to have a minimum number of solutions. We expect that this knowledge can be very profitable in the test data generation, where we want to cover the most critical parts with the lowest number of tests, or in the digitization in teaching where we are always interested in task generation and creation of running examples.

In future work we will increase the problem description by generalizing the number of cubes from 4 to n according to (Demaine et al., 2013). This problem is then called the Cube Stacking Problem and it has been demonstrated, that it is np-complete. This allow us to experimentally evaluate the scalability of our encoding with respect to the number of cubes that we consider.

An other generalization would be to use k-gonal prism pieces and not just cubes for stacking. A study of this generalization is also given in (Demaine et al., 2013) and a CP-encoding will may be useful to solve these instances.

REFERENCES

Aichernig, B. K. and Salas, P. A. P. (2005). Test case generation by OCL mutation and constraint solving. In *Fifth International Conference on Quality Software (QSIC 2005), 19-20 September 2005, Melbourne, Australia*, pages 64–71.

Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004). Boosting systematic search by weighting constraints. In de Mántaras, R. L. and Saitta, L., editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 146–150. IOS Press.

Caballero, R., García-Ruiz, Y., and Sáenz-Pérez, F. (2010). Applying constraint logic programming to SQL test case generation. In *Functional and Logic Programming, 10th International Symposium, FLOPS 2010, Sendai, Japan, April 19-21, 2010. Proceedings*, pages 191–206.

Dechter, R. (2003). *Constraint processing*. Elsevier Morgan Kaufmann.

Degrave, F., Schrijvers, T., and Vanhoof, W. (2009). Towards a framework for constraint-based test case generation. In *Logic-Based Program Synthesis and Transformation, 19th International Symposium, LOPSTR 2009, Coimbra, Portugal, September 2009, Revised Selected Papers*, pages 128–142.

Demaine, E. D., Demaine, M. L., Eisenstat, S., Morgan, T. D., and Uehara, R. (2013). Variations on instant insanity. In *Space-Efficient Data Structures, Streams, and Algorithms*.

Ernsting, M., Majchrzak, T. A., and Kuchen, H. (2012). Dynamic solution of linear constraints for test case generation. In *Sixth International Symposium on Theoret-*

ical Aspects of Software Engineering, TASE 2012, 4-6 July 2012, Beijing, China, pages 271–274.

Hellsten, L., Pesant, G., and van Beek, P. (2004). A domain consistency algorithm for the stretch constraint. In Wallace, M., editor, *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 290–304. Springer.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Lecoutre, C. (2009). *Constraint Networks: Techniques and Algorithms*. Wiley.

López-Ortiz, A., Quimper, C., Tromp, J., and van Beek, P. (2003). A fast and simple algorithm for bounds consistency of the alldifferent constraint. In Gottlob, G. and Walsh, T., editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 245–250. Morgan Kaufmann.

Pesant, G. (2001). A filtering algorithm for the stretch constraint. In Walsh, T., editor, *Principles and Practice of Constraint Programming - CP 2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 183–195. Springer.

Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In Wallace, M., editor, *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer.

Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. `http://www.choco-solver.org/`, last visited 2017-06-20.

Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier, Amsterdam, First edition.

Wotawa, F., Nica, S., and Nica, M. (2011). Debugging and test case generation using constraints and mutations. In *Proceedings of the Ninth Workshop on Intelligent Solutions in Embedded Systems, WISES 2011, Regensburg, Germany, July 7-8, 2011*, pages 95–100.