

Title	Computational complexity and an integer programming model of Shakashaka
Author(s)	Demaine, Erik D.; Okamoto, Yoshio; Uehara, Ryuhei; Uno, Yushi
Citation	IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E97-A(6): 1213-1219
Issue Date	2014-06-01
Type	Journal Article
Text version	publisher
URL	http://hdl.handle.net/10119/12147
Rights	Copyright (C)2014 IEICE. Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno , IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E97-A(6), 2014, 1213-1219. http://www.ieice.org/jpn/trans_online/
Description	



Computational Complexity and an Integer Programming Model of Shakashaka


Erik D. DEMAINÉ^{†a)}, Nonmember, Yoshio OKAMOTO^{††b)}, Ryuhei UEHARA^{†††c)},
and Yushi UNO^{††††d)}, Members

SUMMARY Shakashaka is a pencil-and-paper puzzle proposed by Guten and popularized by the Japanese publisher Nikoli (like Sudoku). We determine the computational complexity by proving that Shakashaka is NP-complete, and furthermore that counting the number of solutions is #P-complete. Next we formulate Shakashaka as an integer-programming (IP) problem, and show that an IP solver can solve every instance from Nikoli’s website within a second.

key words: integer programming, NP-completeness, pencil-and-paper puzzle, Shakashaka

1. Introduction

The puzzle *Shakashaka* is one of many pencil-and-paper puzzles (such as the famous Sudoku) popularized by Japanese publisher Nikoli. Shakashaka was proposed by Guten in 2008, and since then, has become one of the main Nikoli puzzles.

An instance of Shakashaka consists of an $m \times n$ rectangular board of unit squares. Each square is either white or black, and some black squares contain a number. A candidate solution to the puzzle consists of filling in some of the white squares with a black half-square (isosceles right triangle filling half the area) in one of the four ways: . We call such squares *b/w squares*; white squares may also be left entirely white. Each number in a black square specifies the number of b/w squares that should be among four (vertically or horizontally adjacent) neighbors of the black square. (A black square without a number allows any number of b/w neighbors.) The objective of the puzzle is to fill the white squares in the given board while satisfying the above constraints and so that the remaining white area consists only of (empty) squares and rectangles. An example of the puzzle Shakashaka in [1] is shown in Fig. 1(a), and its

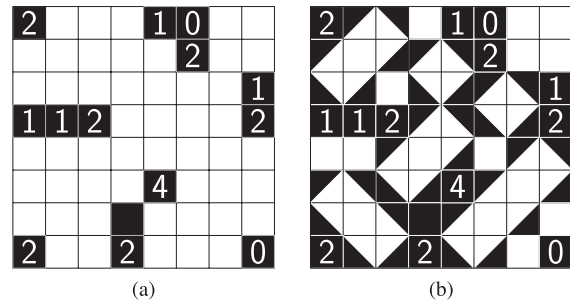


Fig. 1 An instance of the puzzle Shakashaka and its solution ([1]).

(unique) solution is given in Fig. 1(b).

As mentioned in the literature [2], a lot of pencil-and-paper puzzles have been shown NP-complete. However, the computational complexity of Shakashaka has not yet been studied up to the knowledge of the authors. In this paper, we prove that Shakashaka is NP-complete, by a reduction from planar 3SAT. Because our reduction preserves the number of solutions, we also prove that counting the number of solutions to a Shakashaka puzzle is #P-complete.

Next we show how to formulate Shakashaka as a 0-1 integer-programming problem (a linear programming problem in which all variables are restricted to be 0 or 1). Although 0-1 integer programming is one of Karp’s 21 NP-complete problems, there are many efficient solvers from a practical point of view. For example, recent mixed integer-programming solvers run around one billion times faster than those from 1991 [3]. Therefore, once we can formulate a puzzle as a 0-1 integer-programming problem, we can hope to use these solvers to solve the puzzle efficiently in practice.

Some authors have proposed integer-programming formulations of several puzzles before, mainly for the didactic purposes [4]–[8]. The formulation of Shakashaka is not so straightforward because we have to avoid forming nonrectangular orthogonal shapes or nested rectangles. We show that our formulation characterizes the constraints of Shakashaka. We also perform computational experiments, and observe that each instance from Nikoli’s website can be solved within one second by the IP solver SCIP [11].

2. Preliminaries

Let us begin with a formal definition of the puzzle

Manuscript received September 21, 2013.

Manuscript revised January 9, 2014.

[†]The author is with MIT Computer Science and Artificial Intelligence Laboratory, MA 02139, USA.

^{††}The author is with the Graduate School of Informatics and Engineering, The University of Electro-Communications, Chofu-shi, 182-8585 Japan.

^{†††}The author is with the School of Information Science, JAIST, Nomi-shi, 923-1292 Japan.

^{††††}The author is with the Graduate School of Science, Osaka Prefecture University, Sakai-shi, 599-8531 Japan.

a) E-mail: edemaine@mit.edu

b) E-mail: okamotoy@uec.ac.jp

c) E-mail: uehara@jaist.ac.jp

d) E-mail: uno@mi.s.osakafu-u.ac.jp

DOI: 10.1587/transfun.E97.A.1213

Shakashaka. An instance I of *Shakashaka* is a rectangular board of size $m \times n$. Each unit square is colored either *white* or *black*. A black square may contain a number $i \in \{0, \dots, 4\}$. A *solution* of the instance I is a mapping from the set of white squares in I to the set $\{\square, \blacksquare, \blacktriangle, \blacktriangledown, \blacklozenge\}$ satisfying the following conditions:

1. Each white square mapped to \square is left uncolored (white), while each square mapped to $\blacksquare, \blacktriangle, \blacktriangledown,$ or \blacklozenge is colored black and white as indicated (and called a *b/w square*).
2. Each black square that contains the number i has exactly i b/w squares among its four neighbors.
3. Each connected white area forms a white rectangle (or square).

Computationally, *Shakashaka* is a decision problem: for a given instance, does it have a solution? The *counting version* of *Shakashaka* asks to compute the number of distinct solutions to the given instance.

3. NP-Completeness of Shakashaka

In this section, we prove the following theorem:

Theorem 1: *Shakashaka* is NP-complete, even if each black square contains either empty or 1.

The proof is by a reduction from planar 3SAT, one of the well-known NP-complete problems [9]. Let F be an instance of planar 3SAT. That is, F consists of a set $C = \{C_1, C_2, \dots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \dots, x_n\}$, where each clause C_i consists of three literals, and the graph $G = (C \cup \mathcal{V}, \mathcal{E})$ is planar, where \mathcal{E} contains an edge $\{C_i, x_j\}$ if and only if literal x_j or \bar{x}_j is in the clause C_i .

Now we show a reduction from F to an instance I of *Shakashaka*. To do that, we design some gadgets to represent each literal and clause with wire gadget to join them. The key idea is to use the pattern shown in Fig. 2. For the pattern in Fig. 2(a), we have two choices for filling the 2×2

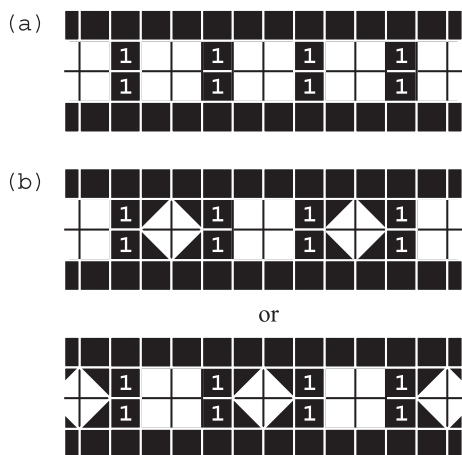


Fig. 2 Basic pattern.

white squares as shown in Fig. 2(b). Essentially, this works as a “wire” to propagate a signal. We regard the 2×2 square containing the four white unit squares in Fig. 2(b) as representing “0,” and the big diamond containing four (different) b/w squares in 2(b) as representing “1.” That is, the “wire” pattern propagates a signal using the parity in two different ways. Using the terminology of [2], we need the gadgets of “variable,” “split,” “corner,” and “clause” to represent F in the form of G , a network of literals and clauses. We describe these gadgets one by one.

(1) Variable gadget:

Figure 3(a)[†] shows the variable gadget. It is easy to see that we have two ways to fill the pattern as in Figs. 3(b), (c). It can propagate its value by the wire gadget as in the figure. It is also easy to obtain the negation of the variable by taking the value at the appropriate position of the wire.

(2) Split gadget/corner gadget:

Figure 4 shows the split and corner gadgets. Using the split gadget, we can increase the degree of the output of a variable gadget.

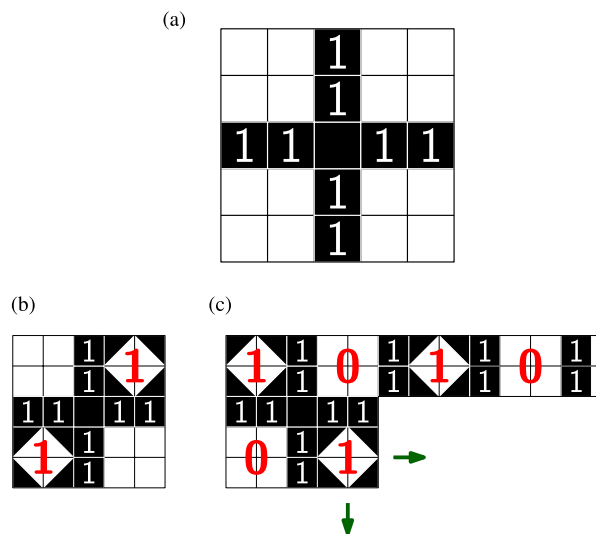


Fig. 3 Variable gadget.

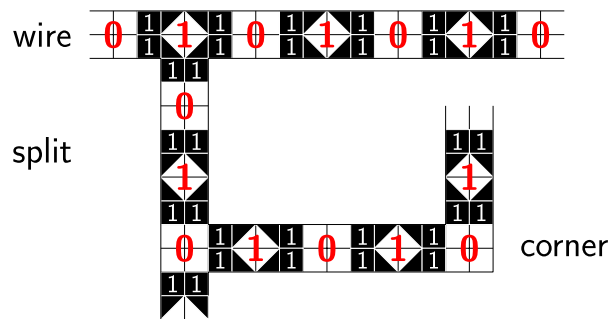


Fig. 4 Split and corner gadgets.

[†]Hereafter, each pattern is assumed to be surrounded by black squares.

(3) Clause gadget:

Figure 5 shows the clause gadget for a clause $C = \{x, y, z\}$. According to the values of x, y, z , we have eight possible cases. Among them, only the case $x = y = z = 0$ violates the condition of Shakashaka (Fig. 6).

The gadgets for wire, variable, split, and corner are aligned properly because they are designed to fit into a 3×3 square tiling. However, at a clause gadget, we have to change the positions of wires to fit the gadget. To shift the position, we use a “parity” gadget shown in Fig. 7(a). Joining copies of the gadget in a straightforward way, we can change the position of a wire arbitrarily (Fig. 7(b)). Since G is planar, we can arrange the variable and clause gadgets on a sufficiently large rectangular board so that they can be joined by edges without crossing. Then we join them by using wire gadgets, split gadgets, and corner gadgets. At a clause gadget, we adjust the positions of wire gadgets by using parity gadgets appropriately. All gaps between gadgets are filled with black empty squares. As a result, we can construct the resulting instance of Shakashaka in a rectangular board of size $O(n) \times O(n)$ in polynomial time of n . An example of a construction of Shakashaka for the instance

$f = C_1 \vee C_2$, where $C_1 = \{x, \bar{y}, w\}$ and $C_2 = \{y, \bar{z}, \bar{w}\}$ is depicted in Fig. 8.

It is easy to see that the resulting Shakashaka has a solution if and only if the original formula F is satisfiable. It is clear that Shakashaka is in the class NP. Therefore, Shakashaka is NP-complete. Moreover, once we fix an assignment of F , the filling pattern of the resulting instance of Shakashaka is uniquely determined. Thus, our reduction is parsimonious, i.e., it preserves the number of solutions. That is, the number of satisfying assignments to the original CNF formula is equal to the number of solutions to the resulting instance of Shakashaka. Because the counting version of planar 3SAT is #P-complete [10], we have the following corollary:

Corollary 2: The counting version of Shakashaka is #P-complete.

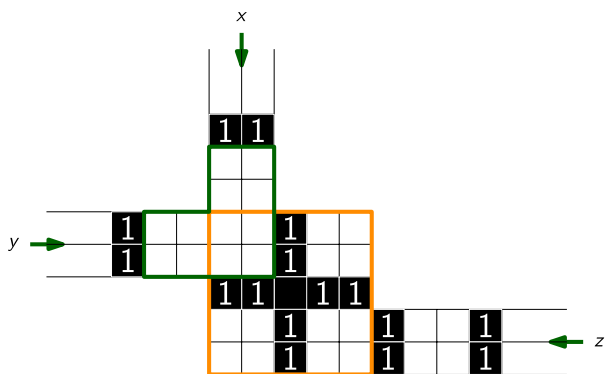


Fig. 5 Clause gadget.

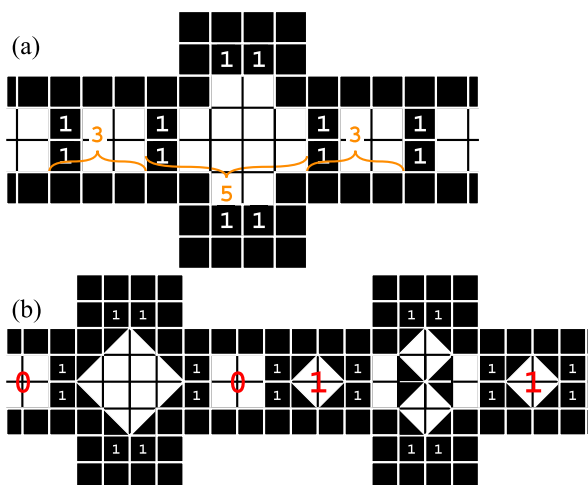


Fig. 7 Parity gadget.

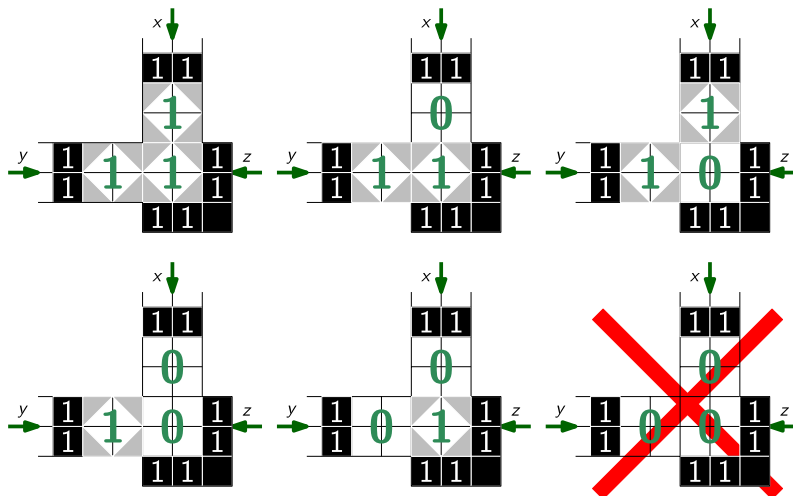


Fig. 6 Feasible cases and infeasible case of a clause gadget.

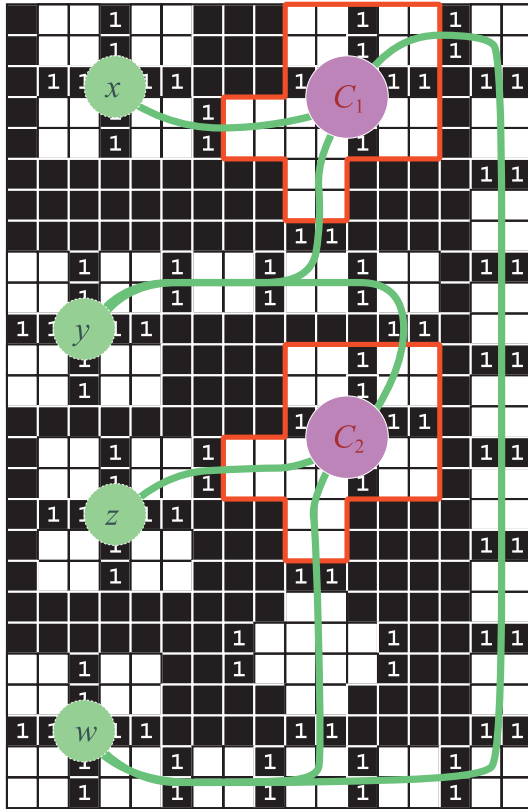


Fig. 8 An example for $C_1 = \{x, \bar{y}, w\}$ and $C_2 = \{y, \bar{z}, \bar{w}\}$.

4. Integer Programming Formulation

We formulate Shakashaka in terms of a 0-1 integer-programming problem. Recall that an instance I of Shakashaka consists of a rectangular board of size $m \times n$. We identify each square by $(i, j) \in \{1, 2, \dots, m\} \times \{1, 2, \dots, n\}$ in the natural way.

(4) Variables:

For each white square (i, j) , we will use five 0-1 variables $x[i, j, \square]$, $x[i, j, \blacktriangle]$, $x[i, j, \blacktriangledown]$, $x[i, j, \blacktriangleleft]$, and $x[i, j, \blacktriangleright]$. Exactly one of these variables has value 1, and the rest are 0, according to the following meaning:

$$\left\{ \begin{array}{l} x[i, j, \square] = 1 \text{ means that } (i, j) \text{ remains white,} \\ x[i, j, \blacktriangle] = 1 \text{ means that } (i, j) \text{ is filled with } \blacktriangle, \\ x[i, j, \blacktriangledown] = 1 \text{ means that } (i, j) \text{ is filled with } \blacktriangledown, \\ x[i, j, \blacktriangleleft] = 1 \text{ means that } (i, j) \text{ is filled with } \blacktriangleleft, \\ x[i, j, \blacktriangleright] = 1 \text{ means that } (i, j) \text{ is filled with } \blacktriangleright. \end{array} \right.$$

We construct a linear system $S(I)$ with the variables $x[i, j, *]$ such that the solutions of the instance I of Shakashaka are in bijection with the solutions of $S(I)$. To this end, we set up five types of linear constraints as described below.

(5) Constraint A (at most one triangle in each white square):

In a solution to I , each white square either remains white, or is filled with one of the four black isosceles right triangles. We map this condition to the following linear equality:

$$x[i, j, \square] + x[i, j, \blacktriangle] + x[i, j, \blacktriangledown] + x[i, j, \blacktriangleleft] + x[i, j, \blacktriangleright] = 1 \tag{1}$$

for each i and j where (i, j) is a white square.

Proposition 3: Let $S_A(I)$ be the linear system that consists of Constraint A. Then any feasible solution of $S_A(I)$ gives the mapping from each white square to exactly one of $\square, \blacktriangle, \blacktriangledown, \blacktriangleleft, \text{ or } \blacktriangleright$.

(6) Constraint B (neighbors of black squares):

Next we look at the black squares (i, j) . First we consider the case that (i, j) contains no number. In this case, (i, j) gives some restrictions to its white neighbors. For example, suppose that $(i - 1, j)$ is white. Then, if $(i - 1, j)$ was \blacktriangle or \blacktriangledown , these two squares make a 45° white corner between them. Thus $(i - 1, j)$ must be $\blacktriangleleft, \blacktriangle, \text{ or } \square$. Hence, in this case, the equation (1) for $(i - 1, j)$ can be replaced by

$$x[i - 1, j, \square] + x[i - 1, j, \blacktriangleleft] + x[i - 1, j, \blacktriangle] = 1 \tag{2}$$

and we can fix $x[i - 1, j, \blacktriangle] = x[i - 1, j, \blacktriangledown] = 0$. For $(i + 1, j)$ and $(i, j \pm 1)$, similar equations and simplifications are applied.

On the other hand, when a black square (i, j) has a number k , it must have k b/w squares as its neighbor. This restriction is described by the following equation:

$$x[i - 1, j, \blacktriangleleft] + x[i - 1, j, \blacktriangle] + x[i + 1, j, \blacktriangleright] + x[i + 1, j, \blacktriangledown] + x[i, j - 1, \blacktriangleleft] + x[i, j - 1, \blacktriangle] + x[i, j + 1, \blacktriangleright] + x[i, j + 1, \blacktriangledown] = k, \tag{3}$$

where $x[i, j, *]$ is regarded as 0 if (i, j) is black. We also fix $x[i - 1, j, \blacktriangle] = x[i - 1, j, \blacktriangledown] = x[i + 1, j, \blacktriangleleft] = x[i + 1, j, \blacktriangleright] = x[i, j - 1, \blacktriangle] = x[i, j - 1, \blacktriangledown] = x[i, j + 1, \blacktriangleleft] = x[i, j + 1, \blacktriangleright] = 0$ to avoid the 45° white angle.

(7) Constraint C (sequences of triangles):

Next we turn to the restrictions to make each connected white area a rectangle. Suppose $x[i, j, \blacktriangleleft] = 1$. In this case, the white triangle at (i, j) can be orthogonal if and only if either $x[i, j + 1, \blacktriangle] = 1$ or $x[i + 1, j + 1, \blacktriangledown] = 1$. Therefore, we obtain the following constraint:

$$x[i, j, \blacktriangleleft] \leq x[i, j + 1, \blacktriangle] + x[i + 1, j + 1, \blacktriangledown]. \tag{4}$$

Moreover, when $x[i, j, \blacktriangleleft] = x[i + 1, j + 1, \blacktriangledown] = 1$, $(i, j + 1)$ must remain white, or $x[i, j + 1, \square] = 1$. (When $(i, j + 1)$ is \blacktriangledown , we have a parity problem; we cannot enclose this area by extending this pattern. The other cases are also prohibited.) This implies the following constraint:

$$x[i, j, \blacksquare] + x[i + 1, j + 1, \blacksquare] \leq x[i, j + 1, \square] + 1. \quad (5)$$

We add the similar constraints for the other directions. Then, we have the following proposition:

Proposition 4: Let $S_C(I)$ be the linear system that consists of Constraints A, B, and C, and fix any feasible solution of $S(I)$. Then, each angle on the boundary of each connected white area given by the mapping is 90° .

(8) Constraint D (exclusion of concave corners):

By Proposition 4, any feasible solution to Constraints A, B, and C produces a pattern consisting of orthogonal white polygons. However, this does not yet exclude concave corners. For example, an instance in Fig. 9(a) has a unique solution in Fig. 9(b). However, up to the constraints A, B, and C, we have another solution in Fig. 9(c). By Equation 4, no b/w square forms a part of a concave corner. Thus, a concave corner may be produced by only white squares. Suppose that $x[i, j, \square] = x[i + 1, j, \square] = x[i, j + 1, \square] = 1$. Then, $(i + 1, j + 1)$ must be \blacksquare or must remain white. Thus we add the following constraints (for all possible directions):

$$x[i, j, \square] + x[i + 1, j, \square] + x[i, j + 1, \square] \leq x[i + 1, j + 1, \square] + x[i + 1, j + 1, \blacksquare] + 2. \quad (6)$$

We now have the following proposition:

Proposition 5: Let $S_D(I)$ be the linear system that consists of Constraints A, B, C, and D, and fix any feasible solution of $S(I)$. Then every connected part of a boundary of a white area is a convex orthogonal polygon, i.e., a rectangle.

(9) Constraint E (Exclusion of Nested White Rectangles):

The last problem is that the linear system so far may produce nested rectangles. (Two rectangles are *nested* if one properly contains another; see Fig. 10. For the instance in

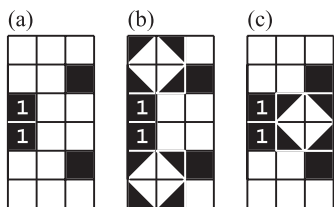


Fig. 9 An instance of the puzzle Shakashaka that allows to have another solution without Constraint D.

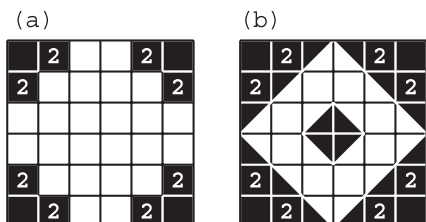


Fig. 10 An instance of the puzzle Shakashaka that allows to have nested rectangles without Constraint E.

Fig. 10(a), the center diamond in Fig. 10(b) is not allowed. But it cannot be avoided by only the constraints from A to D.) We suppose that both of (i, j) and $(i + k, j + k)$ are \blacksquare . Then, to avoid nesting, we must have \blacksquare between them. That is, we must have \blacksquare at $(i + k', j + k')$ for some $0 < k' < k$. And it is not difficult to see that this is a necessary and sufficient condition to avoid nested rectangles. This observation gives us the following constraint:

$$x[i, j, \blacksquare] + x[i + k, j + k, \blacksquare] \leq \sum_{0 < k' < k} x[i + k', j + k', \blacksquare] + 1. \quad (7)$$

Combining all propositions and observations above, we conclude the following:

Theorem 6: Let I be an instance of Shakashaka, and $S(I)$ be the linear system that consists of Constraints A–E. Then, a feasible solution of $S(I)$ gives a solution to I , and vice versa.

5. Experimental Results

In this section, we describe our experimental results. The IP solver we used is SCIP 3.0.0 [11][†] (Binary: Windows/PC, 32 bit, cl 16, intel 12.1: statically linked to SoPlex 1.7.0, Ipopt 3.10.2, CppAD 20120101.3). The machine we used was a laptop (Intel Core2 Duo P8600@2.40 GHz with RAM 4 GB on Windows Vista Business SP2). Each of the ten instances at nikoli.com^{††} was solved in less than one second in our experiments (Table 1). We observed that each of the instances was solved by the presolve functionality of SCIP, and no branch-and-bound procedure was needed. This should be a consequence of a quite powerful presolve in SCIP. With the counting functionality of SCIP, we verified that each of the instances has a unique feasible solution.

We also looked at another instance at nikoli.com, which was prepared for a competition. The board has size 31×45 , the level is Extreme, and the number of white squares is 1230. A solution was obtained in 2.63 seconds.

The other examples are artificial ones (see Fig. 11); for each $n = 1, 2, \dots$, the board of size $2n \times 2n$ consists of $4 \times \sum_{i=1}^{n-1} i = 2n(n - 1)$ black squares, and $4 \times (n - 1)$ black

Table 1 Experimental results for the instances at nikoli.com.

Problem	Size	Level	# of white \square s	Time (sec)
1	10×10	Easy	76	0.02
2	10×10	Easy	77	0.03
3	10×10	Easy	82	0.03
4	10×18	Easy	131	0.07
5	10×18	Medium	156	0.09
6	10×18	Medium	144	0.07
7	14×24	Medium	297	0.21
8	14×24	Hard	295	0.19
9	20×36	Hard	645	0.84
10	20×36	Hard	632	0.91

[†]<http://scip.zib.de/>

^{††}<http://www.nikoli.com/ja/puzzles/shakashaka/>

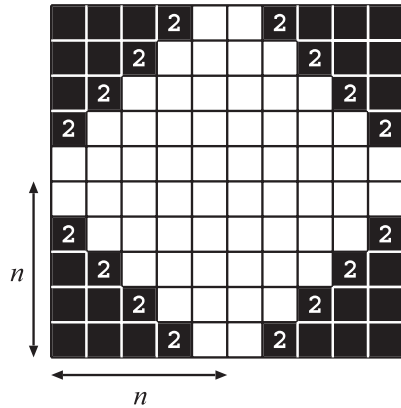


Fig. 11 An artificial example of the puzzle Shakashaka.

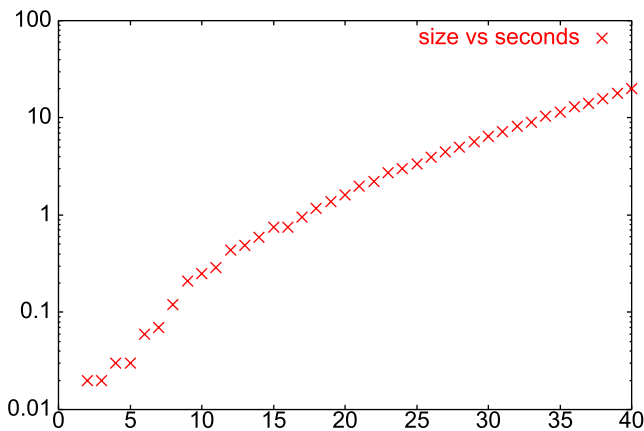


Fig. 12 Seconds for the artificial examples ($n = 2, 3, \dots, 40$).

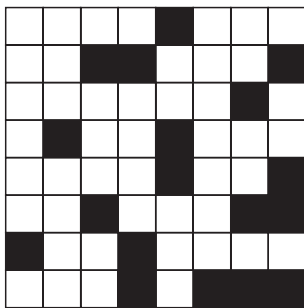


Fig. 13 An instance of Shakashaka without numbers.

squares contain the number 2 as shown in the figure. Each of them has a unique solution. The experimental results for the artificial ones for $n = 2, 3, \dots, 40$ are shown in Fig. 12. For $n = 40$, the solution is obtained in 19.86 seconds. A simple regression shows that the computation time is roughly proportional to 1.18^n .

6. Concluding Remarks

In this paper, we proved that Shakashaka is NP-complete. In our reduction, the black squares contain only the number 1 (or remain blank). An interesting question is to deter-

mine the computational complexity of Shakashaka with no numbers in the black squares. Figure 13 shows a nontrivial example, which has a unique solution. There are two natural questions in this Shakashaka puzzle. How many black squares are required to have a unique solution in an $m \times n$ board? Can this restricted Shakashaka be solved in polynomial time?

References

- [1] Nikoli, Shakashaka 1, vol.151, Pencil and Paper Puzzle Series, Nikoli, Jan. 2012.
- [2] R.A. Hearn and E.D. Demaine, Games, Puzzles, and Computation, A K Peters Ltd., 2009.
- [3] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A.M. Gleixner, S. Heinz, A. Lodi, H.D. Mittelmann, T.K. Ralphs, D. Salvagnin, D.E. Steffy, and K. Wolter, "MIPLIB 2010," Math. Program. Comput., vol.3, no.2, pp.103–163, 2011.
- [4] A. Bartlett, T.P. Chartier, A.N. Langville, and T.D. Rankin, "Integer programming model for the sudoku problem," J. Online Mathematics and its Applications, vol.8, Article ID 1798, 2008.
- [5] R.A. Bosch, "Painting by numbers," Optima, vol.65, pp.16–17, 2001.
- [6] M.J. Chlond, "Classroom exercises in IP modeling: Su doku and the log pile," INFORMS Transactions on Education, vol.5, pp.77–79, 2005.
- [7] W.J.M. Meuffles and D. den Hertog, "Puzzle—Solving the *Battleship* puzzle as an integer programming problem," INFORMS Transactions on Education, vol.10, no.3, pp.156–162, 2010.
- [8] L. Mingote and F. Azevedo, "Colored nonograms: An integer linear programming approach," Proc. EPIA 2009 LNAI, vol.5816, pp.213–224, Springer-Verlag 2009.
- [9] D. Lichtenstein, "Planar formulae and their uses," SIAM J. Comput., vol.11, no.2, pp.329–343, 1982.
- [10] N. Creignou and M. Hermann, "Complexity of generalized satisfiability counting problems," Information and Computation, vol.125, pp.1–12, 1996.
- [11] T. Achterberg, "SCIP: Solving constraint integer programs," Mathematical Programming Computation, vol.1, pp.1–41, 2009.



Erik D. Demaine received a B.Sc. degree from Dalhousie University in 1995, and M.Math. and Ph.D. degrees from University of Waterloo in 1996 and 2001, respectively. Since 2001, he has been a professor in computer science at the Massachusetts Institute of Technology. His research interests range throughout algorithms, from data structures for improving web searches to the geometry of understanding how proteins fold to the computational difficulty of playing games. In 2003, he received a MacArthur Fellowship as a "computational geometer tackling and solving difficult problems related to folding and bending—moving readily between the theoretical and the playful, with a keen eye to revealing the former in the latter". He cowrote a book about the theory of folding, together with Joseph O'Rourke (*Geometric Folding Algorithms*, 2007), and a book about the computational complexity of games, together with Robert Hearn (*Games, Puzzles, and Computation*, 2009).



Yoshio Okamoto was born in 1976. He obtained his Ph.D. from ETH Zurich in 2005, and was an assistant professor at Toyohashi University of Technology from 2005 to 2007, an associate professor at Tokyo Institute of Technology from 2007 to 2010, and a research associate professor at Japan Advanced Institute of Science and Technology from 2010 to 2012. Since 2012, he has been an associate professor at the University of Electro-Communications. His research

focuses on discrete mathematics and theory of computing, especially problems on graphs and discrete geometry.



Ryuhei Uehara received B.E., M.E., and Ph.D. degrees from the University of Electro-Communications, Japan, in 1989, 1991, and 1998, respectively. He was a researcher in CANON Inc. during 1991–1993. In 1993, he joined Tokyo Woman's Christian University as an assistant professor. He was a lecturer during 1998–2001, and an associate professor during 2001–2004 at Komazawa University. He moved to JAIST in 2004, and he is now a professor in School of Information Science. His research in-

terests include computational complexity, algorithms, data structures, and graph algorithms. Especially, he is engrossed in computational origami, games and puzzles from the viewpoints of theoretical computer science. He is a member of EATCS, ACM, and IEEE.



Yushi Uno received the B.E., M.E. and Ph.D. degrees in Engineering from Kyoto University, in 1987, 1989 and 1995, respectively. Currently, he is working at Department of Mathematics and Information Sciences, Graduate School of Science, Osaka Prefecture University, Sakai, 599-8531 Japan. His research interests include combinatorial optimization, algorithmic graph theory, operations research, design and analysis of algorithms. He is a member of Association for Computing Machinery, Operations

Research Society of Japan and Information Processing Society of Japan.