Computational complexity and an integer programming model of Shakashaka

Erik D. Demaine* Yoshio Okamoto† Ryuhei Uehara‡ Yushi Uno§

Abstract

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 if #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.

Keywords: 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 Figure 1(a), and its (unique) solution is given in Figure 1(b).

*Computer Science and Artificial Intelligence Laboratory, MIT, edemaine@mit.edu

†Graduate School of Informatics and Engineering, The University of Electro-Communications (UEC), okamotoy@uec.ac.jp

‡School of Information Science, JAIST, uehara@jaist.ac.jp

§Graduate School of Science, Osaka Prefecture University (OPU), uno@mi.s.osakafu-u.ac.jp

(a)

(b)

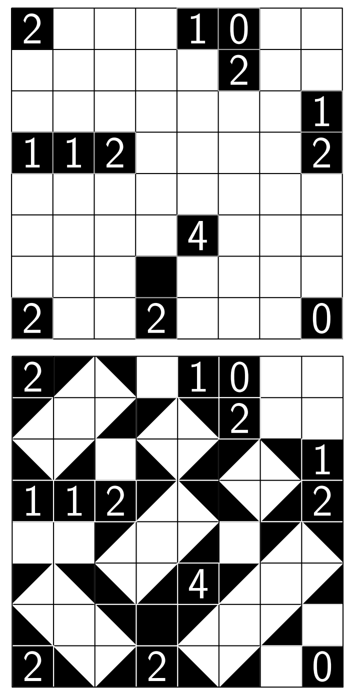

Figure 1: An instance of the puzzle Shakashaka and its solution ([1])

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. 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 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 solvers run around one billion times faster than those from 1991 [3]. Therefore, once we can formulate a puzzle as a 0-1 integer linear 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, 5, 6, 7, 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.

2 Preliminaries

Let us begin with a formal definition of the puzzle 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* $i \in \{0, \ldots, 4\}$. A *solution* of the instance I is a mapping from the set of white squares in I to the set {⊡, ◤, ◥, ◣, ◢} satisfying the following conditions:

1. Each white square mapped to ⊡ is left uncolored (white), while each square mapped to ◤, ◥, ◣, or ◢ 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.*

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 $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$, where each clause C_i consists of three literals, and the graph $G = (\mathcal{C} \cup \mathcal{V}, \mathcal{E})$ 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. The key idea is to use the pattern shown in Figure 2. For the pattern in Figure 2(a), we have two choices for filling the 2×2 white squares as shown in Figure 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 Figure 2(b) as representing "0," and the big diamond containing four (different) b/w squares in 2(b) as representing "1." That is, the

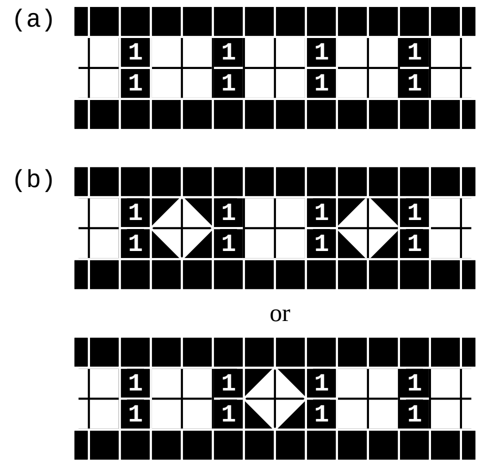

Figure 2: Basic pattern

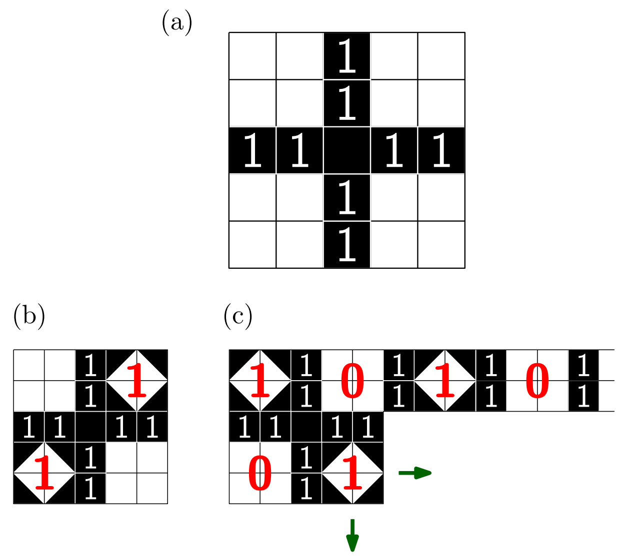

Figure 3: Variable gadget

"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." We describe these gadgets one by one.

Variable gadget: Figure 3(a)[1] shows the variable gadget. It is easy to see that we have two ways to fill the pattern as in Figure 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.

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.

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

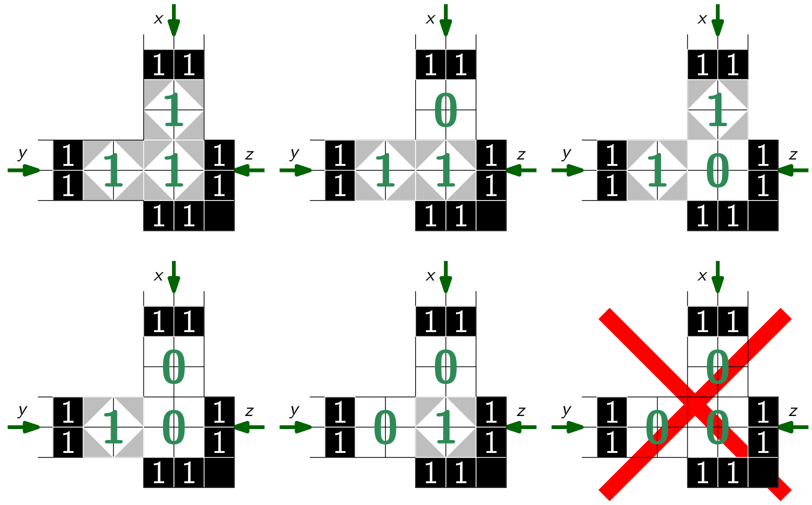

Figure 6: Feasible cases and infeasible case of a clause gadget

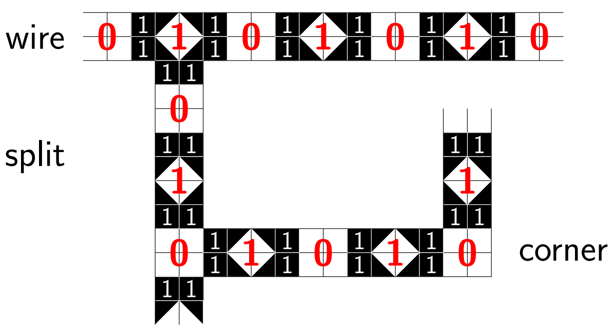

Figure 4: Split and corner gadgets

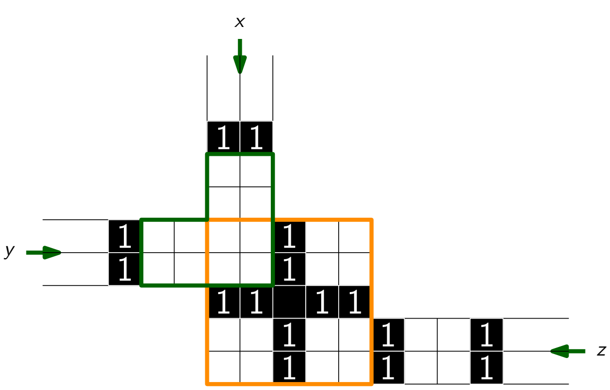

Figure 5: Clause gadget

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 (Figure 6).

The gadgets for wire, variable, split, and corner are aligned properly because they are designed to fit into

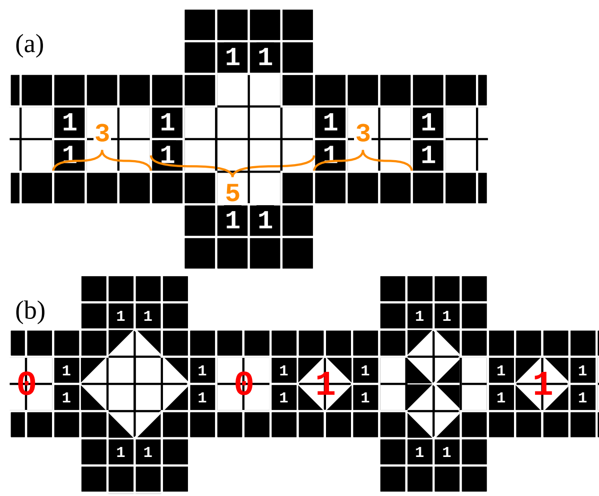

Figure 7: Parity gadget

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 Figure 7(a). Joining copies of the gadget in a straightforward way, we can change the position of a wire arbitrarily (Figure 7(b)). 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 Figure 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 the reduction can be done in polynomial time, and Shakashaka is in the class NP. Therefore, Shakashaka is NP-complete.

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

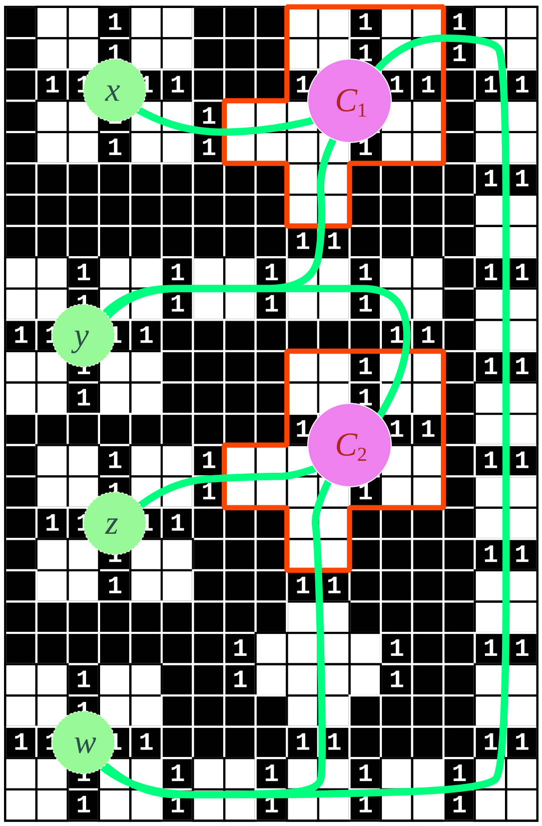

Figure 8: An example for $C_1 = \{x, \bar{y}, w\}$ and $C_2 = \{y, \bar{z}, \bar{w}\}$

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.*

4 Integer Programming Formulation

We formulate Shakashaka in terms of a 0-1 integer program. 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.

Variables: For each white square (i, j), we will use five 0-1 variables $x[i, j, \boxdot]$, $x[i, j, \text{◤}]$, $x[i, j, \text{◥}]$, $x[i, j, \text{◣}]$, and $x[i, j, \text{◢}]$. Exactly one these variables has value 1, and the rest are 0, according to the following meaning:

$$\begin{cases} x[i,j,\boxdot] = 1 & \text{means that } (i,j) \text{ remains white,} \\ x[i,j,\text{◤}] = 1 & \text{means that } (i,j) \text{ is filled with ◤,} \\ x[i,j,\text{◥}] = 1 & \text{means that } (i,j) \text{ is filled with ◥,} \\ x[i,j,\text{◣}] = 1 & \text{means that } (i,j) \text{ is filled with ◣,} \\ x[i,j,\text{◢}] = 1 & \text{means that } (i,j) \text{ is filled with ◢.} \end{cases}$$

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.

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:

$$\begin{aligned} x[i,j,\boxdot] + x[i,j,\text{◤}] + x[i,j,\text{◥}] \\ + x[i,j,\text{◣}] + x[i,j,\text{◢}] = 1 \end{aligned} \tag{1}$$

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 $\boxdot$, ◤, ◥, ◣, or ◢.*

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 $(i-1, j)$ is ◤ or ◥, these two squares make a 45° white corner between them. Thus $(i-1, j)$ must be ◣, ◢, or $\boxdot$. Hence, in this case, the equation (1) for $(i-1, j)$ can be replaced by

$$x[i-1,j,\boxdot] + x[i-1,j,\text{◣}] + x[i-1,j,\text{◢}] = 1 \tag{2}$$

and we can fix $x[i-1,j,\text{◤}] = x[i-1,j,\text{◥}] = 0$.

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:

$$\begin{aligned} x[i-1,j,\text{◣}] + x[i-1,j,\text{◢}] + x[i+1,j,\text{◤}] \\ + x[i+1,j,\text{◥}] + x[i,j-1,\text{◥}] + x[i,j-1,\text{◢}] \\ + x[i,j+1,\text{◤}] + x[i,j+1,\text{◣}] = k, \end{aligned} \tag{3}$$

where $x[i, j, *]$ is regarded as 0 if (i, j) is black. We also fix $x[i-1,j,\text{◤}] = x[i-1,j,\text{◥}] = x[i+1,j,\text{◣}] = x[i+1,j,\text{◢}] = x[i,j-1,\text{◤}] = x[i,j-1,\text{◣}] = x[i,j+1,\text{◥}] = x[i,j+1,\text{◢}] = 0$ to avoid the 45° white angle.

Constraint C (sequences of triangles): Next we turn to the restrictions to make each connected white area a rectangle. Suppose $x[i,j,\text{◣}] = 1$. In this case, the white triangle at (i, j) can be orthogonal if and only if either $x[i,j+1,\text{◢}] = 1$ or $x[i+1,j+1,\text{◣}] = 1$. Therefore, we obtain the following constraint:

$$x[i,j,\text{◣}] \le x[i,j+1,\text{◢}] + x[i+1,j+1,\text{◣}]. \tag{4}$$

Moreover, when $x[i,j,\text{◣}] = x[i{+}1,j{+}1,\text{◣}] = 1$, $(i,j{+}1)$ must remain white, or $x[i,j{+}1,\boxdot] = 1$. (When $(i,j{+}1)$ is ◥, 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:

$$x[i,j,\text{◣}] + x[i+1,j+1,\text{◣}] \le x[i,j+1,\boxdot] + 1. \quad (5)$$

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 $90°$.*

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. 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 $x[i,j,\boxdot] = x[i+1,j,\boxdot] = x[i,j+1,\boxdot] = 1$. Then, $(i+1,j+1)$ must be ◢ or must remain white. Thus we add the following constraints (for all possible directions):

$$\begin{aligned} & x[i,j,\boxdot] + x[i+1,j,\boxdot] + x[i,j+1,\boxdot] \\ \le\ & x[i+1,j+1,\boxdot] + x[i+1,j+1,\text{◢}] + 2. \end{aligned} \quad (6)$$

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.*

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.) We suppose that both of (i,j) and $(i+k,j+k)$ are ◤. Then, to avoid nesting, we must have ◢ between them. That is, we must have ◢ 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:

$$\begin{aligned} & x[i,j,\text{◤}] + x[i+k,j+k,\text{◤}] \\ \le & \sum_{0<k'<k} x[i+k',j+k',\text{◢}] + 1. \end{aligned} \quad (7)$$

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

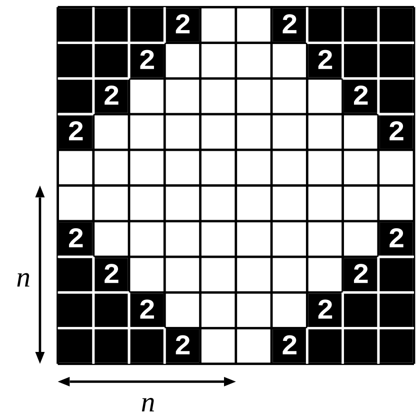

Figure 9: An artificial example of the puzzle Shakashaka.

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][2] (Binary: Windows/PC, 32bit, 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@2.40GHz with RAM 4GB on Windows Vista Business SP2). Each of the ten instances at `nikoli.com`[3] was solved in less than one second in our experiments (Table 1).

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 Figure 9); for each $n = 1, 2, \ldots$, 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 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 $n = 2, 3, \ldots, 40$ are shown in Figure 10. 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

[2] `http://scip.zib.de/`
[3] `http://www.nikoli.com/ja/puzzles/shakashaka/`

Problem	Size	Level	# of white squares	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

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

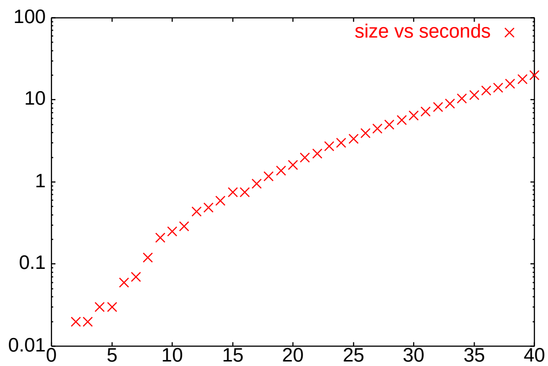

Figure 10: Seconds for the artificial examples ($n = 2, 3, \ldots, 40$).

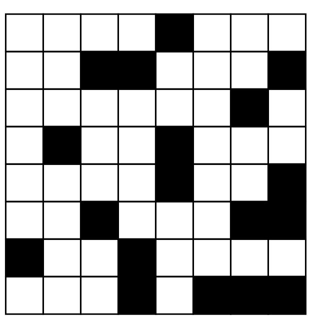

Figure 11: An instance of Shakashaka without numbers

only the number 1 (or remain blank). An interesting question is to determine the computational complexity of Shakashaka with no numbers in the black squares. Figure 11 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. of 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,” Proceedings of EPIA 2009 LNAI vol. 5816, pp.213–224, Springer-Verlag 2009.

[9] D. Lichtenstein, “Planar Formulae and Their Uses,” SIAM J. on Computing, 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.