Electronic Preprint for Journal of Information Processing Vol.23 No.3

Regular Paper

Fillmat is NP-Complete and ASP-Complete

Uejima Akihiro[1,a)] Suzuki Hiroaki[1]

Received: July 31, 2014, Accepted: January 7, 2015

Abstract: We study the computational complexity of a packing puzzle Fillmat, which is a type of pencil-and-paper puzzles made by Japanese puzzle publisher Nikoli. We show that the problem to decide if a given instance of Fillmat has a solution is **NP**-complete by a reduction from the circuit-satisfiability problem (Circuit-SAT). Our reduction is carefully designed so that we can also prove **ASP**-completeness of the another-solution-problem.

Keywords: computational complexity, NP-completeness, ASP-completeness, pencil-and-paper puzzle, Fillmat

1. Introduction and Definitions

A wide variety of puzzles are played all over the world. Pencil-and-paper puzzles are those offered as some figure on the paper and solved by drawing on the figure with a pencil, and such a type of puzzles are popular recreation. Fillmat (or Firumatto), which is considered in this paper, is one of pencil-and-paper puzzles made by Japanese puzzle magazine Nikoli [10].

It seems that one of the sources of fun on playing games and puzzles is their difficulty. From such a viewpoint, the computational complexity of many games and puzzles has been widely studied, and it is known that many commonly played puzzles are **NP**-complete. Hearn and Demaine [4] in 2009 surveyed the complexity results on combinatorial games and puzzles. In recent studies, it has been shown that Hashiwokakero [1], Number Link [8], Kurodoko [7], Shikaku and Ripple Effect [11], Yajilin and Country Road [5], Yosenabe [6], Shakashaka [3] and so on forth, which are pencil-and-paper puzzles published by Nikoli, are **NP**-complete. However, the complexity of Fillmat remained unstudied. Thus, in this paper, we study the computational complexity of the packing puzzle Fillmat defined as follows.

An instance of Fillmat and its solution are shown in **Fig. 1**. Fillmat is played on a rectangular grid P of size $m \times n$, some of whose squares are numbered by $\{1, 2, 3, 4\}$. We call the denoted numbers in P *clues*.

The player's task is to pack 1×1, 1×2, 1×3, 1×4 rectangles into P without overlap under the following four constraints. We call these 4 types of rectangles *blocks*.

(1) Every block may contain at most one clue.
(2) If the block contains a clue, the size of the block must be equal to the number denoted as a clue.
(3) Two blocks of the same size must not be vertically or horizontally adjacent.
(4) The boundary lines of blocks must not form a cross.

Examples that violates the constraints (3) and (4) on Fillmat are shown in **Fig. 2**. We note that P may contain *holes*, i.e., P may have the squares on which a block can not be arranged (see **Fig. 3**). We mention that the original definition of Fillmat does not contain holes, and the original rule on the Nikoli website [10] says that we are to fill every square of the instance.

Throughout this paper, gray squares on a rectangular grid represent holes, and 1×1, 1×2, 1×3, 1×4 blocks are denoted by red, yellow, green, and blue squares, respectively. In this paper, we mainly consider the decision problem of Fillmat defined as follows.

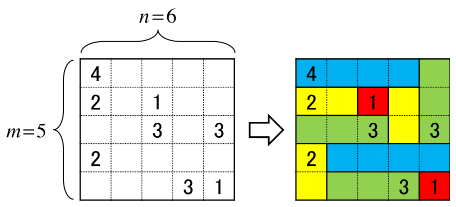


Fig. 1 An instance of Fillmat.

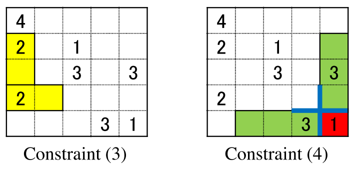


Fig. 2 Examples that violates the constraints (3) and (4) on Fillmat.

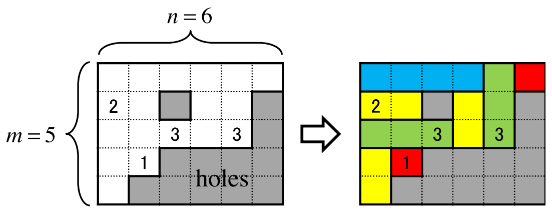


Fig. 3 An instance of Fillmat with holes.

[1] Osaka Electro-Communication University, Neyagawa, Osaka 572–8530, Japan
a) uejima@isc.osakac.ac.jp

Fillmat Decision Problem

Instance:	A rectangular grid P of size $m \times n$, some of whose squares are numbered by $\{1, 2, 3, 4\}$, and which may contain holes,
Question:	Is there an arrangement of blocks in P satisfying the above constraints?

We analyze the computational complexity of the decision problem above and the another-solution-problem of Fillmat, and we show the following theorem in this paper (see Ref. [12] for definitions of a class **ASP** and **ASP**-completeness).

Theorem 1 *Fillmat is* **NP***-complete and* **ASP***-complete.*

2. Proof of Theorem 1

We are now ready to state and prove our claim of **NP**-completeness. It is obvious that the problem to decide whether an instance of Fillmat has a solution is in **NP**, since it can be verified whether a solution candidate is correct in polynomial time.

To prove **NP**-hardness, we construct a reduction from the circuit-satisfiability problem (Circuit-SAT for short), which is the problem of deciding whether given Boolean circuits $C(v_1, v_2, \ldots, v_n)$ with n Boolean variables $v_1, v_2, \ldots, v_n$ have an assignment of the variables that makes the output true. Circuit-SAT has already been shown to be **NP**-complete in Ref. [2], and the proof of **NP**-completeness also implies that the another-solution-problem variant (**ASP** for short) of Circuit-SAT is **ASP**-complete since Cook's reduction is an **ASP** reduction (see Ref. [12] for the definition of the term "**ASP** reduction").

Our reduction is carefully designed so that each solution of Fillmat has a one-to-one correspondence with a solution of the original instance of Circuit-SAT. Therefore, the reduction also implies the result for the another-solution-problem of Fillmat, namely, the **ASP** variant of Fillmat is **ASP**-complete. The **ASP** version of Fillmat is defined as follows: Given an instance P of Fillmat and a solution s, find a solution s' of P other than s.

In the following part, we construct gadgets corresponding to each part of a Boolean circuit by using only the basic logical gates AND and NOT (note that {AND, NOT} is universal), wires, and splits. In the reduction, we must be careful about wires crossing in the circuits. To construct instances of Fillmat from circuits, we consider the Boolean circuits restricted to a plane. For this purpose, we can replace the wires crossing by McColl's planar "cross-over" circuit [9] as shown in **Fig. 4**, which can be constructed by three XOR gates (note that an XOR gate can be expressed as a "planar" circuit consisting of AND and NOT gates).

In our construction of the gadgets, we consider 6×6 squares to be a basic unit, and most of the gadgets consist of a multiple of sizes of a basic unit. In any gadget, the left-side of the gadget receives a Boolean value from the adjacent gadget, and the output value is transmitted from the right-side of the gadget to the next gadget.

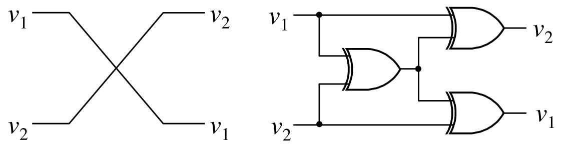


Fig. 4 McColl's planar "cross-over" circuit.

2.1 Input and Wire Gadgets

An input gadget and a wire gadget are shown in **Fig. 5**. We use a basic unit on the left-side of the figure, which is marked by a red dashed line, as an input gadget.

We consider how to place a 1×3 block on a square numbered by 3 in the upper left of an input gadget. From the constraint (4), neither of both sides of a 1×3 block can be located on the square with clue 3. Therefore, we have only two ways of packings as red and blue dashed rectangles illustrated in **Fig. 6**. Note that arrangements of 1×3 blocks on other squares with clue 3 in Fig. 6 are restricted to such two ways.

If a choice of arrangements of a 1×3 block on the upper left square in an input gadget is decided, arrangements of 1×3 blocks on the surrounding squares with clue 3 will be fixed as illustrated in **Fig. 7** from the constraint (3), and 1×1 blocks are placed on the remaining squares. We regard the ways of arrangements (a) and (b) in the figure as truth-value assignments of variables and their transfer: Arrangement (a) corresponds to a true value; in contrast, arrangement (b) corresponds to a false value.

2.2 Split Gadget

A split gadget is shown in **Fig. 8**. This gadget branches a signal wire by combining the same structure as a wire gadget vertically. Note that the restriction of arrangements of 1×3 blocks on squares numbered by 3 holds even if the direction of wire gadgets changes. In addition, wires must bend to connect logical gates in circuits, thus our construction needs to bend wire gadgets to route between our gate-like gadgets. Then, we can use the structure of the split gadget as bend gadgets.

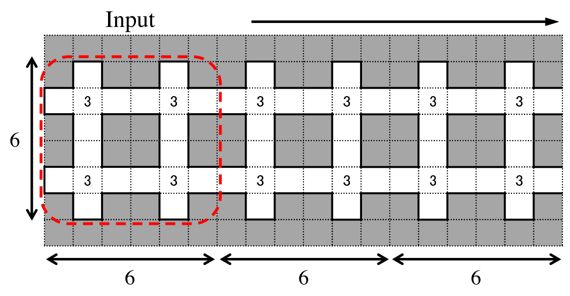


Fig. 5 An input gadget and a wire gadget.

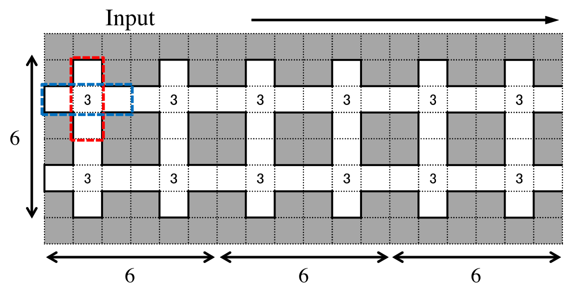
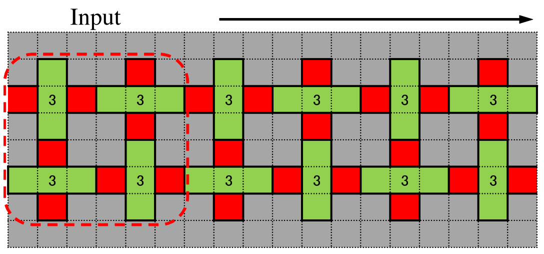
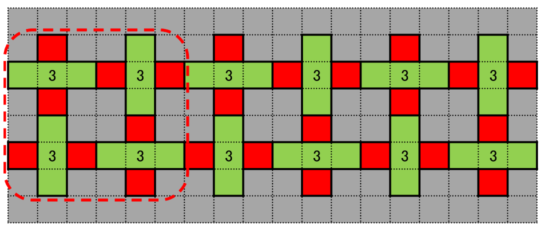


Fig. 6 Arrangement candidates in an input gadget.



(a) True



(b) False

Fig. 7 Truth-value assignments of variables and their transfer. Mention that the assignments (a) and (b) are symmetric, thus Fig. 13 and Fig. 14 in the following are also symmetric.

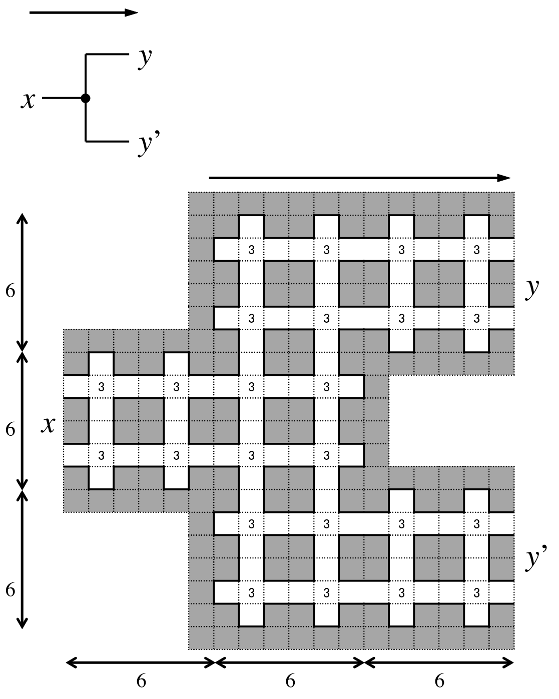


Fig. 8 A split gadget.

2.3 Output Gadget

We use the gadget in **Fig. 9** (a) as an output gadget. The gadget consists of 2×1 squares adding on the lower right of the wire gadget. If the input of the output gadget is true, we have only one way to arrange on the gadget as illustrated in Fig. 9 (b). Otherwise, we have no arrangements satisfying all constraints. For example, the arrangement as Fig. 9 (c) does not satisfy the constraints (3) and (4). Therefore, it has a solution if and only if the value received from the left is true.

2.4 NOT Gadget

Figure 10 shows the NOT gadget. Similar to the case of a wire gadget, arrangements of 1×3 blocks on squares numbered by 3 in the gadget are restricted.

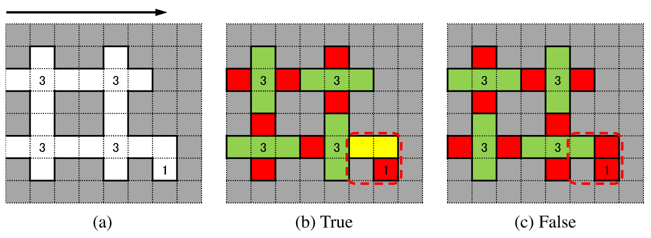


Fig. 9 An output gadget.

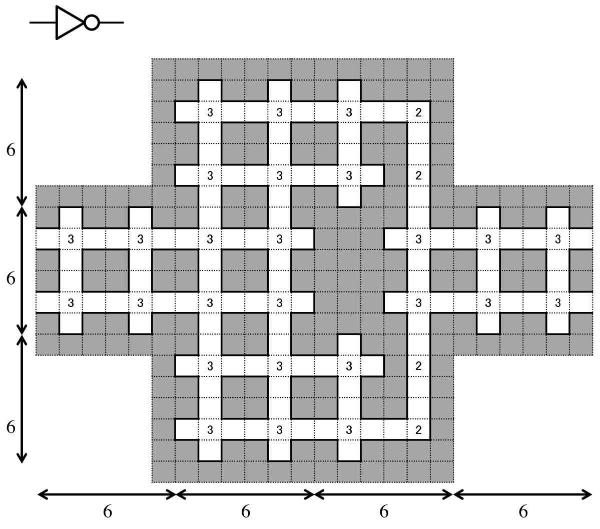


Fig. 10 A NOT gadget.

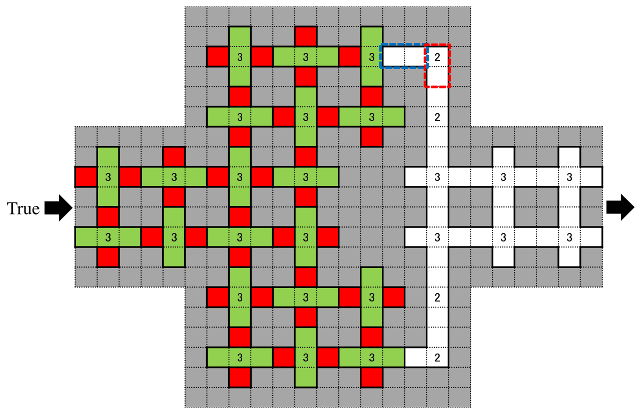


Fig. 11 A NOT gadget with true input.

If the input of the NOT gadget from the left gadget is true, the arrangement on the left half of the gadget is fixed as **Fig. 11** since the structure of the part is the same as a split gadget. Now, we consider how to arrange a 1×2 block on the upper right square with 2. If it is the arrangement like a red dashed rectangle in Fig. 11, we have no feasible arrangements of blocks on the two squares marked by a blue dashed line from the constraint (3). Therefore, the arrangement of a 1×2 block is fixed as **Fig. 12**, and the next 1×2 block is also fixed from the same reason. Similar to the same argument, we have only one way to arrange on the remaining part of the gadget as **Fig. 13**.

If the input of the NOT gadget is false, we have only one way to arrange on the gadget as **Fig. 14** from upper and lower symmetry of the gadget. Therefore, the NOT gadget plays as a NOT gate in a circuit.

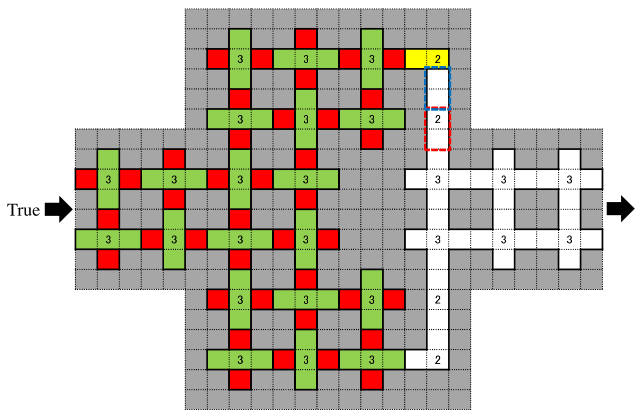


Fig. 12 Arranging 1×2 blocks in a NOT gadget with true input.

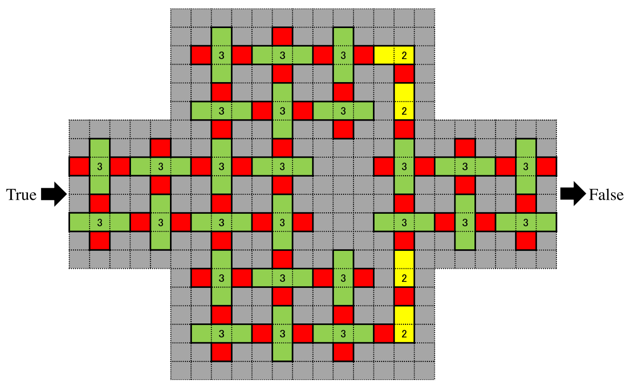


Fig. 13 Solution to a NOT gadget with true input.

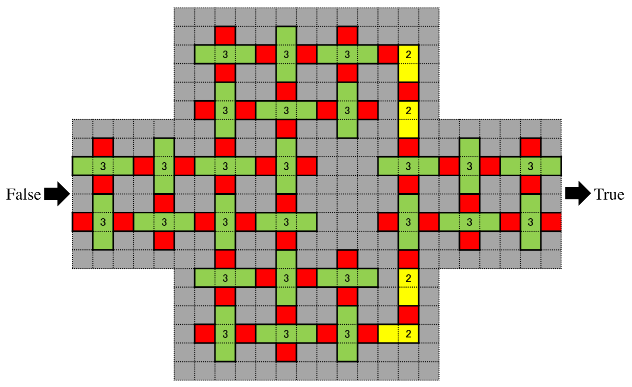


Fig. 14 Solution to a NOT gadget with false input.

2.5 AND Gadget

Figure 15 shows the AND gadget for an AND gate $y = x_1 \wedge x_2$. According to the values of x_1, x_2, we have four possible cases. Arrangements of 1×3 blocks on squares numbered by 3 in the shape of parallel crosses (i.e., in the #-shaped subgrid except for holes) are similarly restricted. Therefore, it is easy to see that the output y is false if either x_1 or x_2 is false, since a horizontally long 1×3 block is placed on either the square B or C (see Figs. 19, 23, and 24).

We consider the case that the inputs (x_1, x_2) of the AND gadget from the left gadget is (true, true) as illustrated in **Fig. 16**. Note that vertically long 1×3 blocks are placed on both squares B and C in this case. Now, it takes notice of arrangement on uncovered squares in the upper side of wire-like gadgets from x_2. Then, we have only one way to arrange a 1×3 block as a red dashed rectangle in Fig. 16. If not, the two 1×3 blocks are vertically ad-

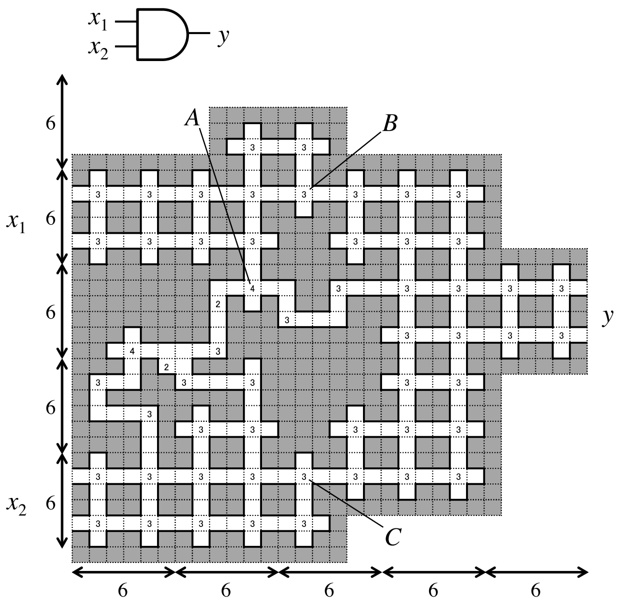


Fig. 15 An AND gadget.

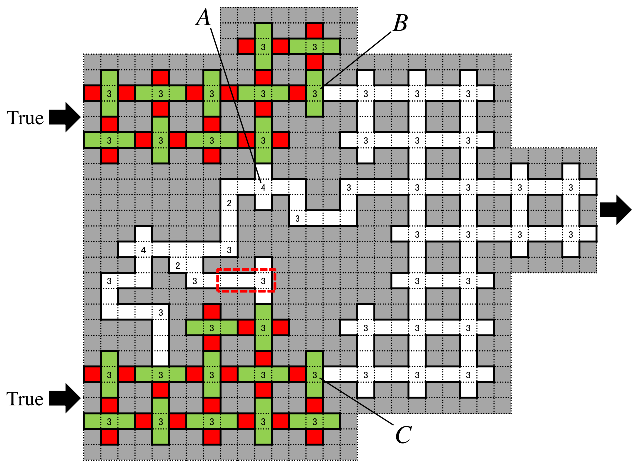


Fig. 16 An AND gadget with (x_1, x_2) = (true, true).

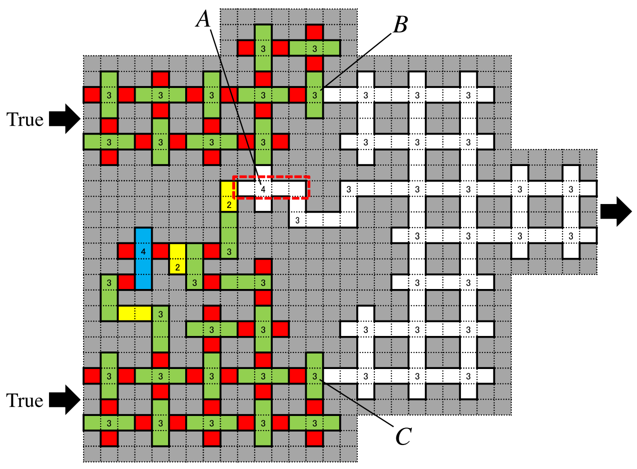


Fig. 17 Arranging blocks in an AND gadget with (x_1, x_2) = (true, true).

jacent and it contradicts the constraint (3). According to such an arrangement, arrangement of blocks on its surrounding squares with numbers is fixed and the remaining squares are covered by 1×1 blocks as illustrated in **Fig. 17**.

After that, we must place a 1×4 block on the square A as a red

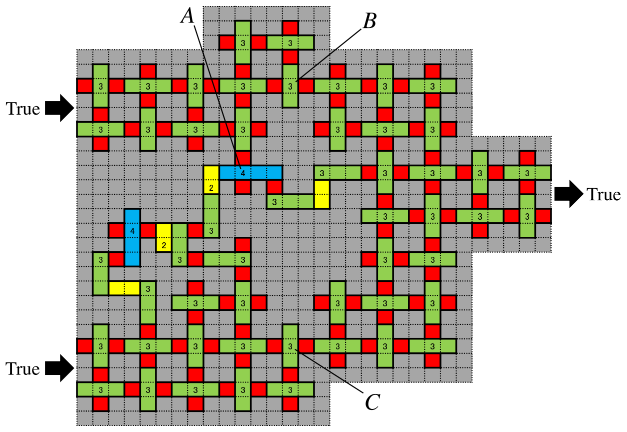


Fig. 18 Unique solution to an AND gadget with (x_1, x_2) = (true, true).

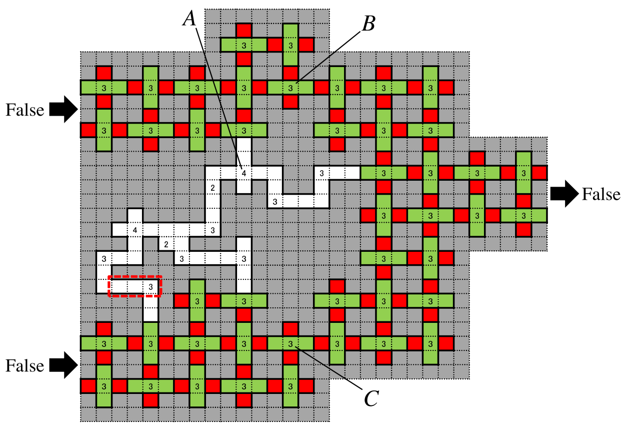


Fig. 19 An AND gadget with (x_1, x_2) = (false, false).

dashed rectangle in Fig. 17, thus two 1×3 blocks are arranged by flush right on the right-side squares with clue 3, and it yields that the remaining squares are covered by 1×1 and 1×2 blocks as illustrated in **Fig. 18**. Therefore, pushing by the 1×3 block on the part, we have only one arrangement on the right-side of AND gadget. Consequently, it requires that the right adjacent gadget receives the value true when both inputs are true.

When the inputs (x_1, x_2) of the AND gadget from the left gadget is (false, false), the arrangement of blocks on the shape of parallel crosses including blocks with clue 3 is fixed as illustrated in **Fig. 19**. Then, we have only one way to arrange a 1×3 block as a red dashed rectangle in Fig. 19. According to such an arrangement, the arrangement of blocks on its surrounding squares with clues is fixed and the remaining squares are covered by 1×1 and 1×2 blocks as **Fig. 20**. If we place a 1×3 block like a red dashed rectangle in Fig. 20, there exists no feasible arrangements of blocks on the two squares marked by a blue dashed line. Therefore, the arrangement of a 1×3 block is fixed as **Fig. 21**, and the next 1×2 block is also fixed from the same reason.

Now, we consider how to arrange a 1×4 block on the square with clue 4 near the center of the gadget. If we place the block like a red dashed rectangle in Fig. 21, there exists no feasible arrangements of blocks on the two squares marked by a blue dashed line. Therefore, the arrangement of blocks on its surrounding squares is fixed as illustrated in **Fig. 22**. Note that if the horizontally long 1×4 block is arranged by flush right, it contradicts the constraint (4) on the bottom left corner of the block. This arrangement cor-

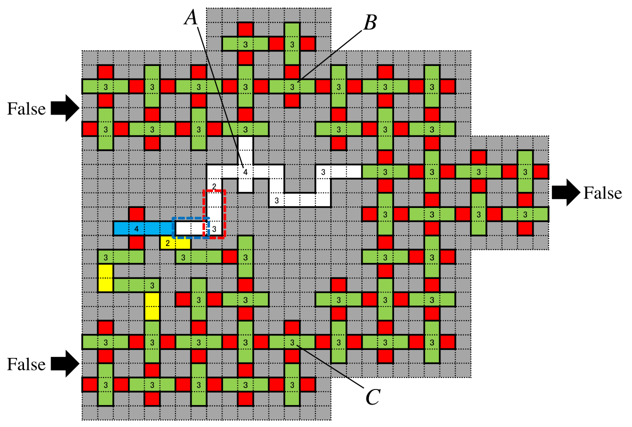


Fig. 20 Arranging blocks in an AND gadget with (x_1, x_2) = (false, false).

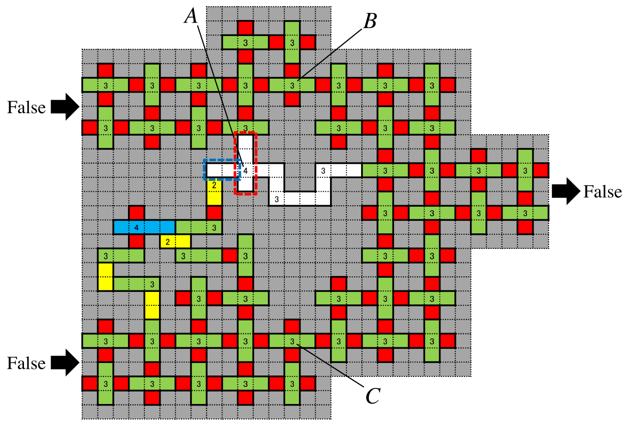


Fig. 21 Arranging a 1×4 block in an AND gadget with (x_1, x_2) = (false, false).

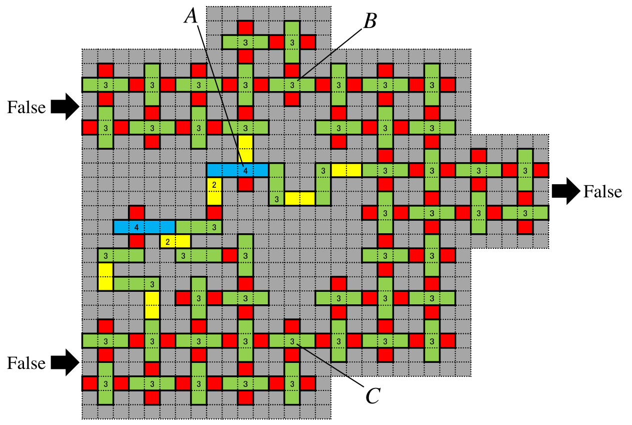


Fig. 22 Unique solution to an AND gadget with (x_1, x_2) = (false, false).

responds to the behavior of an AND gate with (false, false) as both inputs.

We consider the case that the inputs (x_1, x_2) of the AND gadget from the left gadget is (true, false). Similar to the same argument for using the case (x_1, x_2) = (false, false), we have only one way to arrange on the remaining part of the gadget as illustrated in **Fig. 23**. Note that two squares on the right-side of square B must be covered by a 1×2 block as Fig. 23.

When the inputs (x_1, x_2) of the AND gadget from the left gadget is (false, true), the arrangement on most of the squares is fixed as **Fig. 24**. In fact, the arrangement of blocks on the shape of parallel crosses including blocks with clue 3 is fixed by false

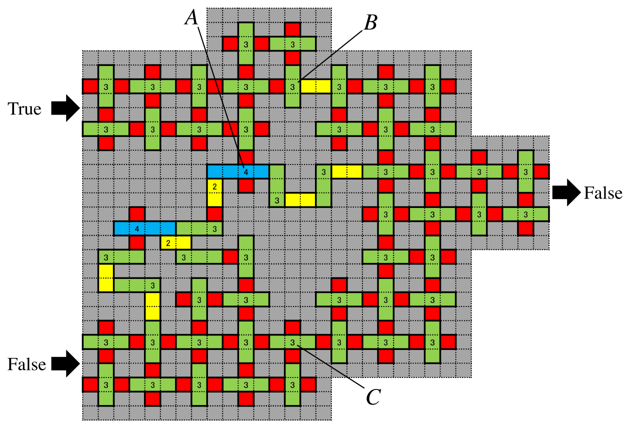


Fig. 23 Unique solution to an AND gadget with (x_1, x_2) = (true, false).

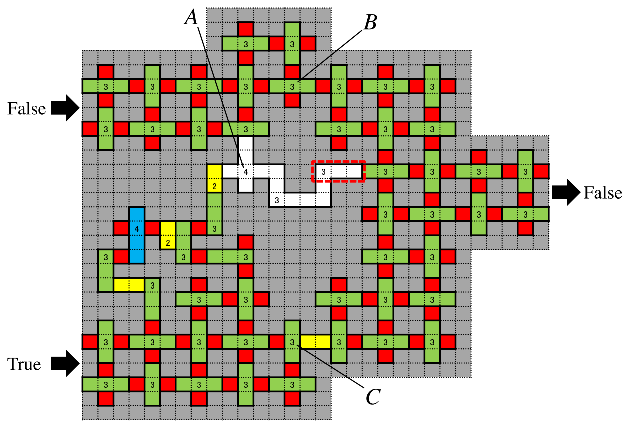


Fig. 24 An AND gadget with (x_1, x_2) = (false, true).

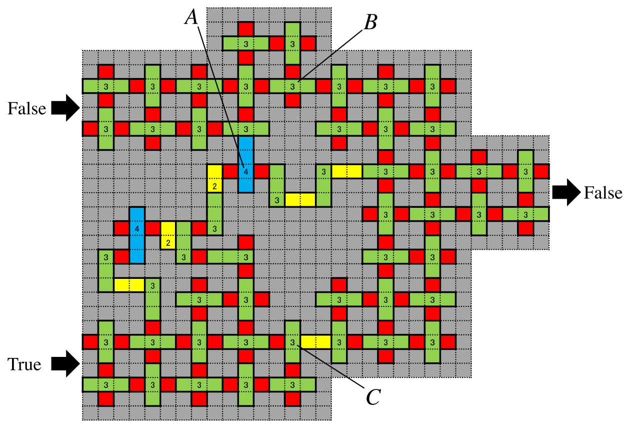


Fig. 25 Unique solution to an AND gadget with (x_1, x_2) = (true, true).

value x_1, moreover, we have only one way to arrange most of the squares in the upper side of wire gadgets of x_2 from the same argument for using the case (x_1, x_2) = (true, true).

Arrangement on the remaining squares is considered. Since the arrangement like a red dashed rectangle in Fig. 24 does not satisfy the constraint (3), blocks placed on the remaining squares are fixed as illustrated in **Fig. 25**.

Therefore, the AND gadget in Fillmat instance corresponds to an AND gate in a circuit. Moreover, there is no other way of arranging blocks on the AND gadget for each case. Thus, feasible arrangements on the AND gadget have a one-to-one correspondence with the behavior of an AND gate.

3. Proof of Correctness of the Reduction

All the necessary gadgets we listed are constructed in the previous section. The area of the grid P where no gadget exists is filled by holes. As a result, we can construct the resulting instance of Fillmat in polynomial time of the size of the original Boolean circuit C.

It is easy to see that the resulting instance of Fillmat has a solution if and only if the original circuit has an assignment of the variables that makes the output true. Namely, the instance of Fillmat obtained by the reduction correctly simulates a Boolean circuit. Therefore, Fillmat Decision Problem is **NP**-complete. Moreover, once we fix an assignment of the variables of C, the arranging pattern of the resulting instance of Fillmat is uniquely determined. That is, arrangements of blocks on P of Fillmat have a one-to-one correspondence with the behavior of the original Boolean circuit C. Therefore, **ASP** version of Fillmat is **ASP**-complete.

4. Conclusions

In this paper, we have studied the computational complexity of Fillmat and we proved that Fillmat is **NP**-complete and **ASP**-complete by reducing the Circuit-SAT problem to Fillmat. The resulting instance of Fillmat contains holes, hence the complexity of Fillmat without holes is still open. In our reduction, the squares with clue 4 appear only in AND gadgets. An interesting question is to determine the computational complexity of Fillmat without the number 4 as a clue.

Acknowledgments This research was supported in part by Institute of Informatics, Osaka Electro-Communication University.

References

[1] Andersson, D.: Hashiwokakero is NP-complete, *Inf. Process. Lett.*, Vol.109, No.19, pp.1145–1146 (2009).
[2] Cook, C.S.: The complexity of theorem proving procedures, *3rd ACM Symposium on Theory of Computing*, pp.151–158 (1971).
[3] Demiane, E.D., Okamoto, Y., Uehara, R. and Uno, Y.: Computational Complexity and an Integer Programming Model of Shakashaka, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Science*, Vol.E97-A, No.6, pp.1213–1219 (2014).
[4] Hearn, R.A. and Demaine, E.D.: *Game, Puzzles, & Computation*, A.K. Peters Ltd., MA, USA (2009).
[5] Ishibashi, A., Sato, Y. and Iwata, S.: NP-completeness of Two Pensil Puzzles: Yajilin and Country Road, *Utilitas Mathematica*, Vol.88, pp.237–246 (2012).
[6] Iwamoto, C.: Yosenabe is NP-complete, *Journal of Information Processing*, Vol.22, No.1, pp.40–43 (2014).
[7] Kölker, J.: Kurodoko is NP-complete, *Journal of Information Processing*, Vol.20, No.3, pp.694–706 (2012).
[8] Kotsuma, K. and Takenaga, Y.: NP-completeness and Enumeration of Number Link Puzzle, *IEICE Technical Report*, Vol.109, No.465, pp.1–7 (2010) (in Japanese).
[9] McColl, W.: Planar crossovers, *IEEE Trans. Comput.*, Vol.30, No.2, pp.223–225 (1981).
[10] Nikoli: Rules of Fillmat (online), available from ⟨http://www.nikoli.co.jp/ja/puzzles/fillmat.html⟩ (accessed 2014-08-30).
[11] Yakenaga, Y., Aoyagi, S., Iwata, S. and Kasai, T.: Shikaku and Ripple Effect are NP-Complete, *Congressus Numerantium*, Vol.216, pp.119–127 (2013).
[12] Yato, T. and Seta, T.: Complexity and Completeness of Finding Another Solution and its Application to Puzzles, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E86-A, No.5, pp.1052–1060 (2003).

Uejima Akihiro was born in 1975. He received a B.E. and M.E. from Information Systems Engineering, Department of Information and Computer Sciences, Toyohashi University of Technology in 1998 and 2000, respectively, and Dr. of Informatics degree from Department of Communications and Computer Engineering, Graduate School of Informatics at Kyoto University in 2005. He was a lecturer during 2005–2013, and has been an associate professor since 2013 in Department of Engineering Informatics, Osaka Electro-Communication University. His research interest is in graph theory, computational complexity. He is a member of IEICE, IPSJ, the Operations Research Society of Japan, the Language and Automaton Symposium.

Suzuki Hiroaki was born in 1991. He received a B.Sc. from Department of Engineering Informatics, Faculty of Information and Communication Engineering, Osaka Electro-Communication University in 2014. From 2014, he has been a student of Master course in Division of Information and Computer Science, Graduate School of Engineering at Osaka Electro-Communication University.