Computational complexity of numberless Shakashaka

Aviv Adler[*] Michael Biro[†] Erik Demaine[*] Mikhail Rudoy [‡] Christiane Schmidt[§]

Abstract

Shakashaka, like Sudoku, is a pencil-and-paper puzzle. In this paper we show that Shakashaka is NP-complete in the case of numberless black squares.

1 Introduction

Shakashaka is a pencil-and-paper puzzle, proposed by Guten in 2008 and popularized by the Japanese publisher Nikoli [1].

An instance of Shakashaka consists of an $m \times n$ rectangle of unit squares. Initially, each square is colored either black or white, and black squares may also contain an integer between 0 and 4, inclusive. The solver proceeds by filling in the initially white squares with squares consisting of a black and a white triangle in one of four orientations: ◩ ⬔ ◪ ⬕. We denote these collectively as *b/w squares*. The white squares may also be left blank. In addition, the numbers written in black squares constrain the solver by specifying the number of b/w squares that must neighbor the given square (in its four vertically and horizontally neighboring squares).

An instance is considered *solved* if every maximal connected white region on the board is a rectangle (axis-aligned or rotated by 45°) and each numbered black square has exactly as many b/w square neighbors as is specified by its number. For an example and its (unique) solution, refer to Figure 1.

Demaine et al. [3] proved that Shakashaka is NP-complete. They used a reduction from planar 3-SAT, and the black squares in the reduction either contained the number 1 or remained blank. In addition, they showed that Shakashaka can be formulated as a 0-1-integer program and gave experiments using IP-solvers (namely SCIP 3.0.0) to solve instances of sizes up to 20×36. Two questions remained in the concluding re-

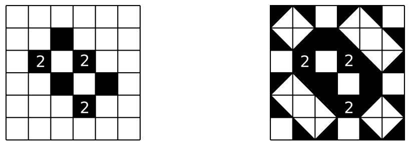


Figure 1: (a) An instance of Shakashaka and (b) its unique solution.

marks of this paper, one concerned settling the computational complexity of Shakashaka without numbers in the black squares.

In this paper we answer this question and show that Shakashaka without numbers in the black squares is NP-complete by a reduction from POSITIVE PLANAR 1-IN-3 SAT, a variant of the well known PLANAR 3-SAT problem, shown to be NP-complete by Mulzer and Rote [5]. The reduction is parsimonious, and, hence, also shows #P-completeness. We also include an easier, but non-parsimonious, reduction from PLANAR 3-SAT, which is well-known to be NP-complete.

2 NP-completeness of numberless Shakashaka

Definition 1 *An instance F of the POSITIVE PLANAR 1-IN-3 SAT problem is a Boolean formula in 3-CNF: it 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 variables ("POSITIVE" indicates that no negated variables appear in the C_i's). Moreover, the variable-clause incidence graph $G = (\mathcal{C} \cup \mathcal{V}, E)$ is planar, where $\{C_i, x_j\} \in E \Leftrightarrow x_j$ is in C_i. It is sufficient to consider formulae where G has a rectilinear embedding, see Knuth and Raghunathan [4]. The POSITIVE PLANAR 1-IN-3 SAT problem is to decide whether there exists a truth assignment to the variables such that exactly one variable in each clause is true.*

Theorem 1 *Shakashaka without numbers in the black squares is NP-complete.*

Proof. [via Positive Planar 1-in-3 SAT]

The proof is by reduction from POSITIVE PLANAR 1-IN-3 SAT, which was shown to be NP-complete by Mulzer and Rote [5]. Let F be an instance of the POSITIVE PLANAR 1-IN-3 SAT problem. We turn the rectilinear embedding of G into a Shakashaka board: we present the variables, clauses and edges by pieces of

[*]Computer Science and Artificial Intelligence Laboratory, MIT, Massachusetts, USA. Email: `adlera@mit.edu`, `edemaine@mit.edu`.

[†]Department of Mathematics and Statistics, Swarthmore College, USA. Email: `mbiro1@swarthmore.edu`.

[‡]Electrical Engineering and Computer Science department and Mathematics department, MIT, Massachusetts, USA. Email: `mrudoy@mit.edu`.

[§]The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel. Email: `cschmidt@cs.huji.ac.il`. Supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

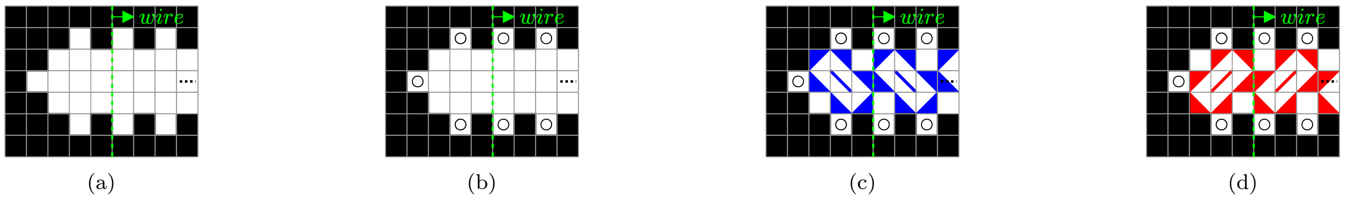


Figure 2: (a) The variable gadget, (b) with enforced white pixels and (c),(d) the two possible feasible solutions. We associate the "kite" in blue with a truth setting of "false" and the "kite" in red with a truth setting of "true".

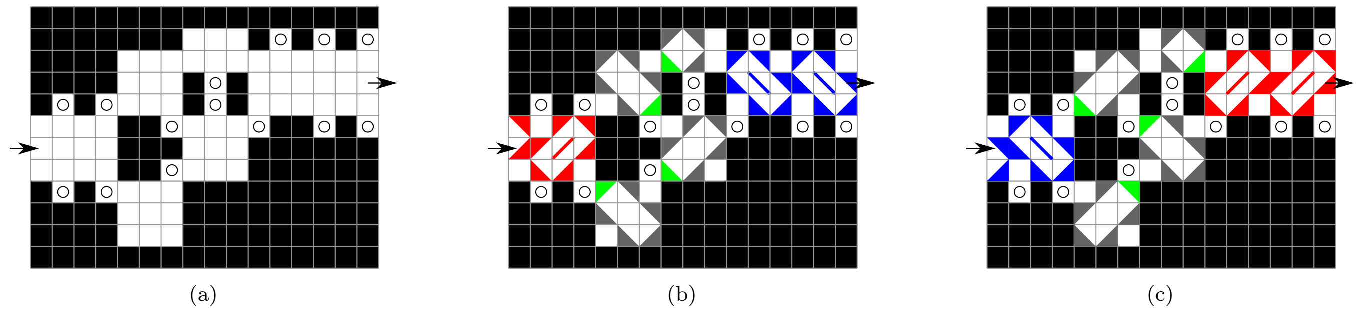


Figure 3: (a) The NOT gadget. (b),(c) The wires connected by the NOT gadget always satisfy opposite truth assignments. In (b) the gadget is entered with a truth assignment corresponding to "true" and left with a truth assignment corresponding to "false". Those roles are reversed in (c). Some enforced triangles are shown in green to facilitate understanding.

the board that need to be filled in. We will give the details of the gadgets in the following.

The **variable gadget** is shown in Figure 2(a). The empty circles in Figure 2(b) indicate (by the construction) enforced white pixels. There exist exactly two feasible solutions for the variable gadget, shown in blue and red in Figures 2(c) and (d) and corresponding to a truth setting of "false" and "true", respectively. In both cases we use a "kite", a sloped structure, occupying 7 out of the pixels of a 3×3-square. For the blue solution, the kite is oriented from top left to bottom right, for the red solution it is oriented from top right to bottom left (both indicated by a line in the rectangle's center).

The initial truth value is propagated by a **wire gadget** as indicated in Figure 2.

Parity. Note that, by construction, the kites propagating through the wires (and all other gadgets below) do so at regular intervals of three squares in both the up/down and left/right directions; i.e., the kites that propagate the truth assignments each fit inside a 3×3-square. In fact, the gadgets of our construction force these kites to be placed inside the tiles of a single 3×3-tiling of the plane. This ensures that the gadgets can be constructed and the kites will align without any shifting.

The **NOT gadget**, shown in Figure 3, enables us to reverse the truth assignment in a variable wire. The NOT gadget will be used in the bend gadget and split gadget as a black box, and is only used there.

The **bend gadget**, shown in Figure 4, enables us to bend a wire to match bends in the rectilinear embedding of G while enforcing that the same truth assignments continue to propagate along the wire.

The **split gadget**, shown in Figure 5, enables us to increase the number of wires propagating the truth assignment of a variable gadget.

The **at-most gadget**, shown in Figure 6, enables us to enforce that at most one of a pair of truth assignments is true. It admits a feasible Shakashaka board in all cases except for two true inputs, in which case an infeasible Shakashaka board is obtained.

The related **at-least gadget**, shown in Figure 7, enables us to enforce that at least one of a pair of truth assignments is true. It admits a feasible Shakashaka board in all cases except for two false inputs, in which case an infeasible Shakashaka board is obtained.

The **"XOR" gadget**, shown in Figure 8, takes two

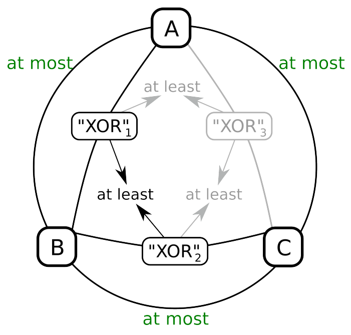
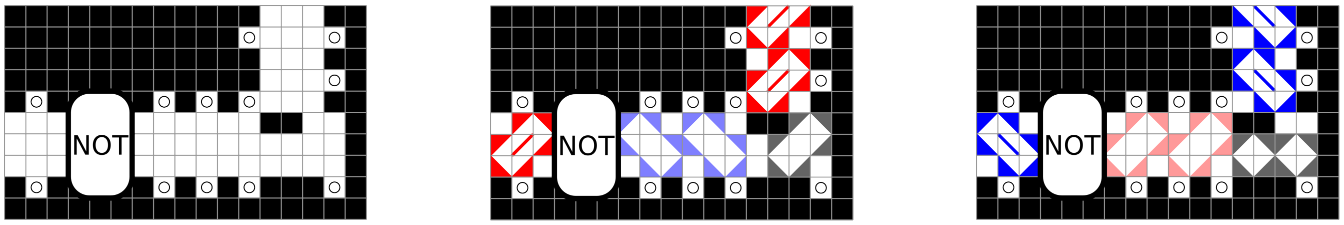
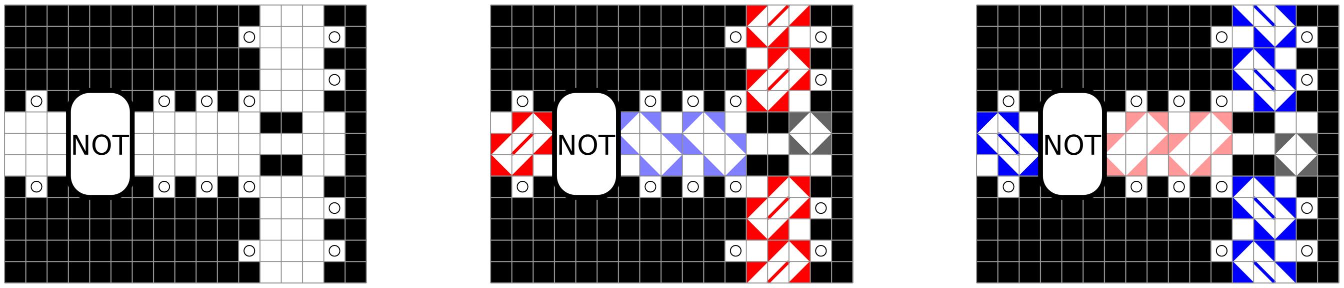


Figure 9: The clause gadget. The gray components ensure that the reduction is parsimonious.



(a) (b) (c)

Figure 4: (a) The bend gadget. (b),(c) The wires connected by the bend always satisfy the same truth assignment.



(a) (b) (c)

Figure 5: (a) A split of the corridor. (b),(c) The wires connected by the split always satisfy the same truth assignment.

wires as input and outputs:

false/false	$\rightarrow$	false
false/true	$\rightarrow$	true or false possible
true/false	$\rightarrow$	true or false possible
true/true	$\rightarrow$	infeasible.

Finally, the **clause gadget**, shown in Figure 9, enforces that exactly one of the three variables included in the clause is set to true. Three variables, represented by A, B and C in Figure 9, are pairwise combined by the at-most gadget. This combination can only be solved if there is at most one true variable among A, B, and C (i.e. the possibilities are false/false/false, true/false/false, false/true/false, and false/false/true). Consequently, we only need to exclude the false/false/false case. We combine each of two pairs of variables with an "XOR" gadget ("XOR"$_1$ and "XOR"$_2$) and combine the results in the at-least gadget. Note that the "XOR" gadgets would yield an infeasible Shakashaka board for two true inputs, but this case has already been excluded.

If all variables are set to false, both "XOR" gadgets must output false. The subsequent combination of the two "XOR" outputs with an at-least gadget results in an infeasible Shakashaka board. If one variable is set to true, at least one "XOR" gadget can output true. Therefore, the subsequent combination of the two "XOR" outputs with an at-least gadget is possible and does not render the board infeasible.

Thus, the resulting Shakashaka has a solution if and only if exactly one variable per clause is set to true, that is, if and only if the original POSITIVE PLANAR 1-IN-3 SAT formula F is satisfiable. It is easy to see, that this reduction is possible in polynomial time. Moreover, given a filled in board it is easy to check whether it constitutes a feasible Shakashaka solution, hence, Shakashaka is in the class NP. Consequently, Shakashaka without numbers in the black squares is NP-complete. So, this gadget, i.e., everything except the gray part in Figure 9, yields the Theorem's statement. But, we want to obtain a parsimonious reduction: once we fix an assignment of F, the filling pattern of the resulting Shakashaka instance is uniquely determined.

Given that the only possible combinations are the possibilities are false/false/false, true/false/false, false/true/false, and false/false/true, one of the "XOR"$_i$ has input false/false, hence, it has to output false. The other two "XOR"$_j$, "XOR"$_k$ ($i \neq j \neq k, i, j, k \in \{1, 2, 3\}$) have input true/false or false/true, i.e., they output either true or false. But then we combine all pairs of "XOR" outputs with an at-least gadget: "XOR"$_i$ outputs false, thus, each of "XOR"$_j$ and "XOR"$_k$ must output true to obtain a feasible board. Hence, we have a one-to-one correspondence between solutions to F and the resulting Shakashaka instance, i.e., the reduction is parsimonious. $\square$

Because the counting version of POSITIVE PLANAR 1-IN-3 SAT is #P-complete [2], we have:

Corollary 1 *The counting version of Shakashaka is #P-complete.*

Definition 2 *An instance F of the PLANAR 3-SAT problem is a Boolean formula in 3-CNF consisting of a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$. Clauses in F contain variables*

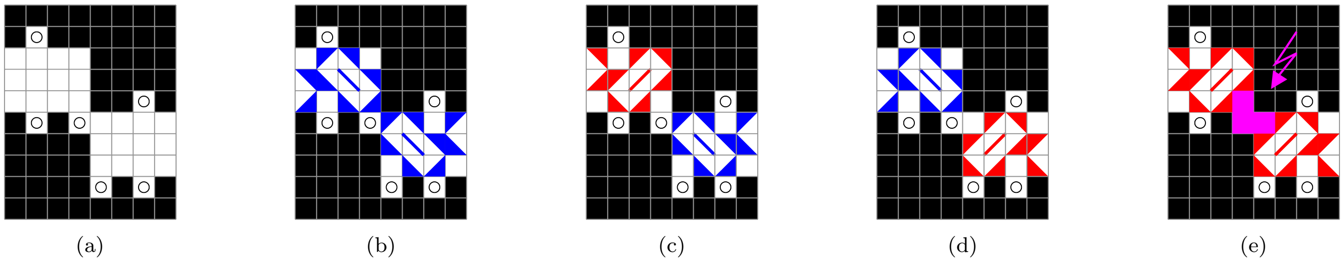


Figure 6: (a) The "at most" gadget. (b) with two false inputs, (c)/(d) with one true and one false input, (e) with two true inputs the board cannot be completed.

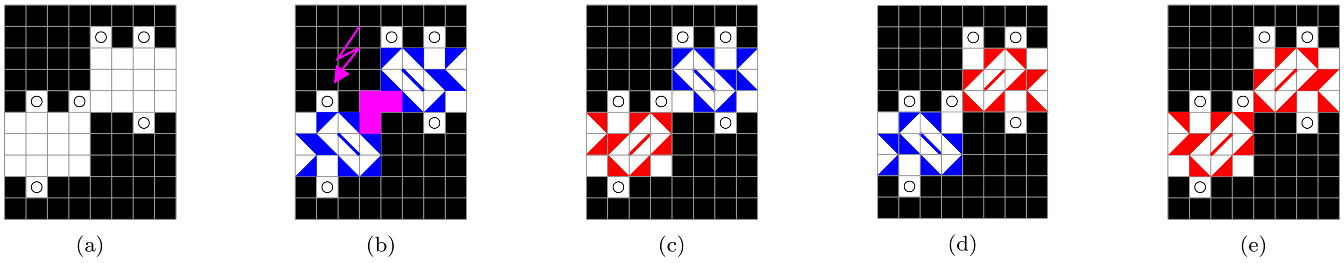


Figure 7: (a) The "at least" gadget: (b) with two false inputs the board cannot be completed, (c)/(d) with one true and one false input, (e) with two true inputs.

and negated variables, denoted as literals *(e.g. 'x_1' or '$\neg x_7$'). A clause is satisfied if and only if it contains at least one* `true` *literal, and the formula F is true if and only if all its clauses are satisfied. The variable-clause incidence graph G is planar and it is sufficient to consider formulae where G has a rectilinear embedding.*

We now present the alternate reduction.

Proof. [via Planar 3-SAT]

In this proof, we give a reduction from PLANAR 3-SAT. We turn the rectilinear embedding of G into a Shakashaka board, much along the lines of the previous proof. We will first discuss the basic representation of variables in this reduction, and then show how this can be used to build wires (which are very easy to split and negate) and clauses.

The basic variable gadget is a 2×3 rectangle, which can be filled in 3 ways, as shown in Figure 10; by looping it as shown in Figure 11, we can eliminate the trivial solution of Fig. 10(a), giving the two solutions shown in Fig. 10(b),(c) (the orientations of the 2×2 groups in each 2×3 block must match because otherwise there will be an 'L'-shaped white tetromino). We can set a given loop of this kind to represent each variable x_i; the variable is set to `true` if the 2×2 half-filled group is in the clockwise direction, and `false` if the 2×2 half-filled group is in the counterclockwise direction. There cannot be any other solutions because the edge shared between any pair of 2×3 blocks is only 1 wide, and hence no diagonally-oriented white rectangles can fit through (otherwise they would need to have a width of less than 1 which is impossible in Shakashaka). As long as our attachments are only 1 tile wide, this will prevent white rectangles from bridging two adjacent blocks and ensure that the only solutions possible are those described here.

There are 3 different patterns of loops, as shown in Figure 12, so as to allow maximum flexibility in our wire construction (shown in Figure 13); these all satisfy the property of having exactly two solutions, which differ in the orientation of the 2×2 groups. Wires are built by attaching loops together to form chains; each chain consists of a series of loops, each adjacent pair of which shares a 2×3 block; in each chain is a special 'variable' loop, which is where the variable setting is read. We note that these loops alternate between having the 2×2 group in the clockwise direction and in the counterclockwise direction; we refer to a loop as being 'synced'

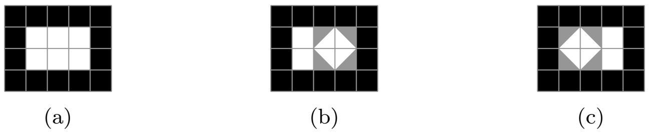


Figure 10: Basic variable gadget with possible solutions.

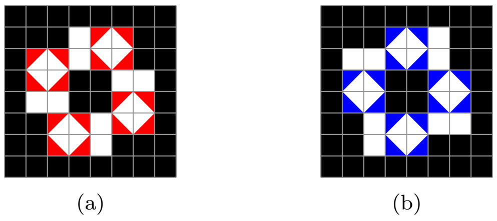


Figure 11: Variable loop with a truth setting of (a) true (red) and (b) false (blue).

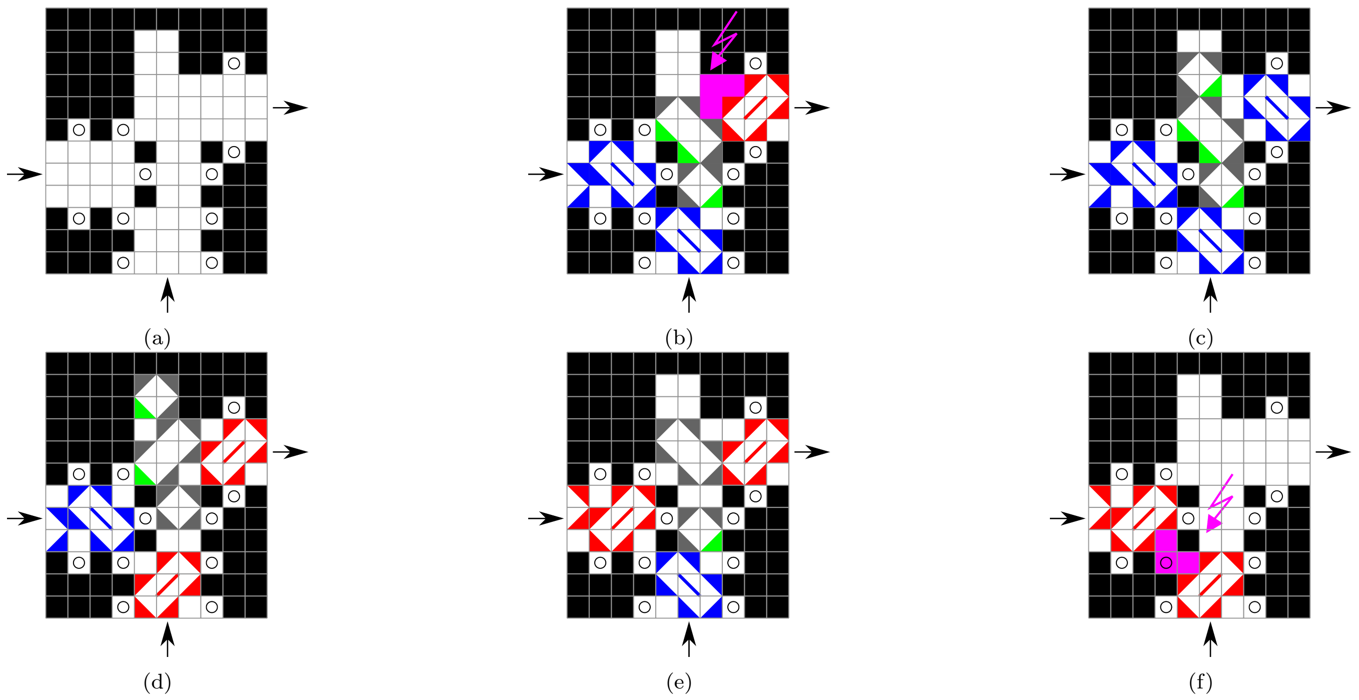


Figure 8: (a) The "XOR" gadget. (b)/(c) two false inputs cannot be completed for a true output (infeasible Shakashaka board indicated in purple), but may be completed for a false output. (d)/(e) both true/false false/true combinations allow a true output, (f) two inputs of true result in an infeasible Shakashaka board. Enforced triangles are shown in green.

if it shares its orientation with the variable loop (i.e. if it is an even distance away in the chain); otherwise it is 'de-synced' and has the opposite orientation. Thus, once the 'variable' loop contained in the chain is set, every other loop within the chain is forced into a particular orientation depending on whether it is synced with the variable loop, as sketched in Figure 13.

We also note that it is very easy to bend a chain (by attaching the next loop to one of the 2×3 blocks to the side rather than to the one opposite to the previously-shared 2×3 block) and to split a chain (by simply attaching two loops to one in a T-junction). As before, the parity of the distance of each loop in the chain from the variable loop determines whether it is synced or de-synced.

The clause gadget is as shown in Figure 14(a); each of the three 2×3 "input" blocks (denoted a, b, and c) is attached to a corresponding variable chain. The attachment occurs at a synced loop if the literal represented is not negated, and at a de-synced loop if the literal represented is negated (i.e. to represent 'x_i' we attach to a synced loop, and to represent '$\neg x_i$' we attach at a de-synced loop). Because of the multiple loop patterns, by increasing the scale of the board by a constant factor we can allow space for the chains to correct the offset and allow the correct parity loop to be in the given spot. If the literal is attached to any of the three 'input' blocks is `false`, the attached loop will be in the `false` state, thus forcing the 2×2 group in the 2×3 'input' block to be placed away from the main body of the clause gadget; if the literal is `true` then there is a choice of where

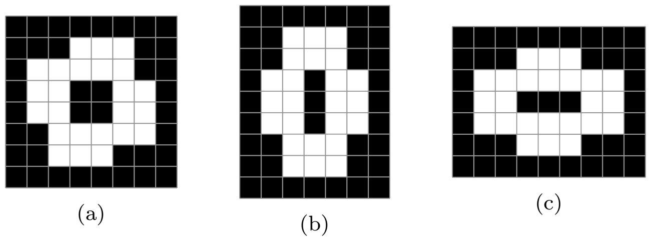


Figure 12: The different patterns for variable loops.

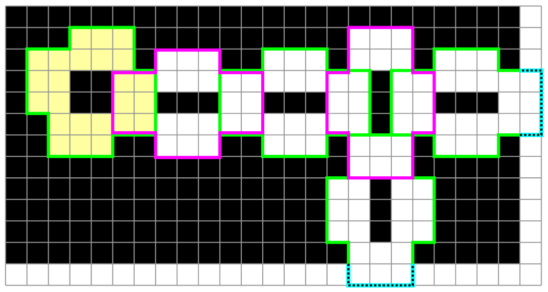


Figure 13: The variable loop is shaded yellow. Synced loops are indicated in green, de-synced loops in pink. Chain continuations are shown in turquoise; note that by using the different attachment points to a loop, bends and splits can be achieved (as shown).

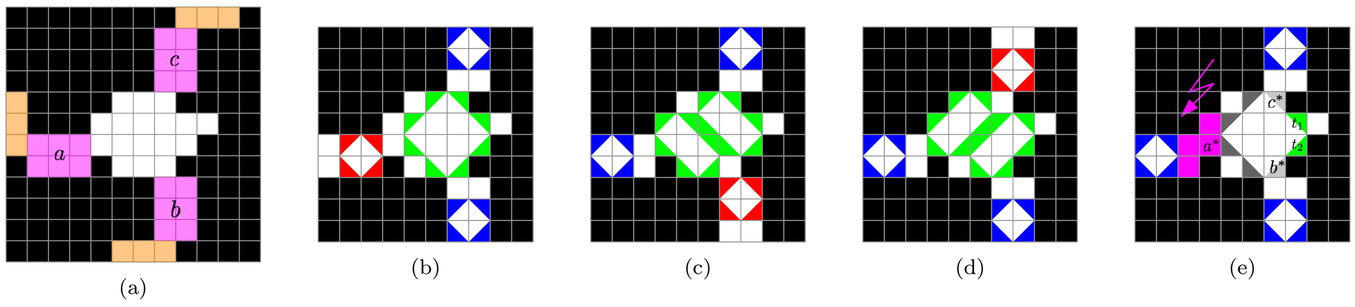


Figure 14: (a) The clause gadget. The input blocks (a, b, c) are indicated in pink, the chains feeding those blocks in orange. Note that the clause still works if input block a is rotated to be vertical (so in panel (e), the disallowed pink region would be a rotated 'L'); this allows extra flexibility in connecting the clauses up to the chains. (b)-(d) The board for one true (red) and two false (blue) literals. (e) For three false literals (blue) the board cannot be completed.

the 2×2 group can be (either towards or away).

To get the clause gadgets to conform to the format given by Knuth and Raghunathan [4], we can wind the chain connecting at the top of the gadget over the right side of the clause's main body (with two 90° bends), such that the three chains come in from the bottom.

We now need to show the following:

1. If at least one of the three literals is `true`, the clause can be satisfied (there is a solution in the gadget).

2. If all of the three literals are `false` then there is no solution within the clause (thus preventing the whole Shakashaka instance from being solved).

This would mean that the Shakashaka board generated has a solution if and only if the formula F is satisfiable.

We note that since a `true` literal can be made to mimic a `false` literal (by allowing placement of the 2×2 group away from the body of the clause), we only need to show that the clause is satisfiable if exactly one of the three literals is `true`; this is because if more than one is `true`, we can have one of the satisfied literals act as `true` and the others mimic `false`. This allows us to handle (1) by Figure 14(b)-(d).

The second result is then shown by the following, as depicted in Figure 14(e). Since each literal is `false` in this case, we are forced to put all three 2×2 groups away from the main body of the clause. The tiles highlighted in green (t_1 and t_2) are forced to be filled in the given way. This is because if t_2 is left blank, then t_1 cannot be left blank as it would create a non-rectangular white shape; but t_1 in this case cannot be filled either, as each of the four orientations of fill result in a non-right angle, again violating the rectangular shape constraint. We then consider the tiles adjacent to the three 'input' blocks; we refer to them as a^*, b^*, and c^* (which are next to inputs a, b, and c respectively). None of these can be left blank (or else there is 270° angle, which is not allowed). However, b^* and c^* can only be filled in the manner shown, as any other orientation of fill will also result in a non-right angle. This forces the white rectangle formed (in part) by tiles t_1, t_2, b^*, and c^* to be closed as shown. But this means that a^* cannot be filled, thus proving that no solution is possible.

Hence, a clause gadget can be solved in Shakashaka if and only if the clause in F that it represents is satisfiable. Thus, we can conclude that the Shakashaka board generated from F via this reduction (which can easily be seen to be polynomial-time) is solvable if and only if all clauses in F can be simultaneously satisfied, i.e. if and only if F is satisfiable. This completes the reduction, showing that Shakashaka is NP-hard (and by virtue of polynomial-time verification, as discussed in the previous proof, it is therefore NP-complete). □

3 Conclusion

In this paper we showed that Shakashaka without numbers in the black squares is NP-complete.

In the future, we like to address the second question from the paper by Demaine et al. [3]: given an $m \times n$ board, what is the minimum number k of black squares that is necessary to obtain a board with a unique solution? Another natural question asks for this number if the black squares contain numbers.

References

[1] http://www.nikoli.co.jp/en/puzzles/shakashaka.html, NIKOLI Co., Ltd. Accessed January 5, 2014.

[2] E. D. Demaine. Lecture 15, 6.890. http://courses.csail.mit.edu/6.890/fall14/lectures/L15.html.

[3] E. D. Demaine, Y. Okamoto, R. Uehara, and Y. Uno. Computational complexity and an integer programming model of Shakashaka. In *Proc. 25th Canad. Conf. Comput. Geom., CCCG 2013*. Carleton University, Ottawa, Canada, 2013.

[4] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal of Discrete Math.*, 5(3):422–427, 1992.

[5] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2), 2008.