Tatamibari is NP-complete

Aviv Adler
Massachusetts Institute of Technology, Cambridge, MA, USA
adlera@mit.edu

Jeffrey Bosboom
Massachusetts Institute of Technology, Cambridge, MA, USA
jbosboom@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
edemaine@mit.edu

Martin L. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
mdemaine@mit.edu

Quanquan C. Liu
Massachusetts Institute of Technology, Cambridge, MA, USA
quanquan@mit.edu

Jayson Lynch
Massachusetts Institute of Technology, Cambridge, MA, USA
jaysonl@mit.edu

Abstract

In the Nikoli pencil-and-paper game Tatamibari, a puzzle consists of an $m \times n$ grid of cells, where each cell possibly contains a clue among $\boxplus$, $\boxminus$, $\boxed{|}$. The goal is to partition the grid into disjoint rectangles, where every rectangle contains exactly one clue, rectangles containing $\boxplus$ are square, rectangles containing $\boxminus$ are strictly longer horizontally than vertically, rectangles containing $\boxed{|}$ are strictly longer vertically than horizontally, and no four rectangles share a corner. We prove this puzzle NP-complete, establishing a Nikoli gap of 16 years. Along the way, we introduce a gadget framework for proving hardness of similar puzzles involving area coverage, and show that it applies to an existing NP-hardness proof for Spiral Galaxies. We also present a mathematical puzzle font for Tatamibari.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Nikoli puzzles, NP-hardness, rectangle covering

Digital Object Identifier 10.4230/LIPIcs.FUN.2020.1

Related Version This paper is also available on arXiv at https://arXiv.org/abs/2003.08331.

1 Introduction

Nikoli is perhaps the world leading publisher of pencil-and-paper logic puzzles, having invented and/or popularized hundreds of different puzzles through their *Puzzle Communication Nikoli* magazine and hundreds of books. Their English website [29] currently lists 38 puzzle types, while their "omopa list" [28] currently lists 456 puzzle types and their corresponding first appearance in the magazine.

Nikoli's puzzles have drawn extensive interest by theoretical computer scientists (including the FUN community): whenever a new puzzle type gets released, researchers tackle its computational complexity. For example, the following puzzles are all NP-complete: Bag / Corral [13], Country Road [20], Fillomino [31], Hashiwokakero [8], Heyawake [19], Hiroimono

/ Goishi Hiroi [7], Hitori [17, Section 9.2], Kakuro / Cross Sum [32], Kurodoko [22], Light Up / Akari [25], LITS [26], Masyu / Pearl [14], Nonogram / Paint By Numbers [30], Numberlink [23, 2], Nurikabe [24, 18], Shakashaka [12, 3], Slitherlink [32, 31, 1], Spiral Galaxies / Tentai Show [15], Sudoku [32, 31], Yajilin [20], and Yosenabe [21].

Allen et al. [5] defined the ***Nikoli gap*** to be the amount of time between the first publication of a Nikoli puzzle and a hardness result for that puzzle type. They observed that, while early Nikoli puzzles have a gap of 10–20 years, puzzles released within the past ten years tend to have a gap of < 5 years.

In this paper, we prove NP-completeness of a Nikoli puzzle first published in 2004 [27] (according to [28]), establishing a Nikoli gap of 16 years.[1] Specifically, we prove NP-completeness of the Nikoli puzzle ***Tatamibari*** (タタミバリ), named after Japanese tatami mats. A Tatamibari ***puzzle*** consists of a rectangular $m \times n$ grid of unit-square cells, some k of which contain one of three different kinds of clues: ⊞, ◫, and ⊟. (The remaining $m \cdot n - k$ cells are empty, i.e., contain no clue.) A ***solution*** to such a puzzle is a set of k grid-aligned rectangles satisfying the following constraints:

1. The rectangles are disjoint.
2. The rectangles together cover all cells of the puzzle.
3. Each rectangle contains exactly one symbol in it.
4. The rectangle containing a ⊞ ("square") symbol is a square, i.e., has equal width (horizontal dimension) and height (vertical dimension).
5. The rectangle containing a ⊟ ("horizontal") symbol has greater width than height.
6. The rectangle containing a ◫ ("vertical") symbol has greater height than width.
7. No four rectangles share the same corner (***four-corner constraint***).

To prove our hardness result, we first introduce in Section 2 a general "gadget area hardness framework" for arguing about (assemblies of) local gadgets whose logical behavior is characterized by area coverage. Then we apply this framework to prove Tatamibari NP-hard in Section 3. In Appendix A, we show that our framework applies to at least one existing NP-hardness proof, for the Nikoli puzzle Spiral Galaxies [15].

We also present in Section 4 a mathematical puzzle font [11] for Tatamibari, consisting of 26 Tatamibari puzzles whose solutions draw each letter of the alphabet. This font enables writing secret messages, such as the one in Figure 1, that can be decoded by solving the Tatamibari puzzles. This font complements a similar font for another Nikoli puzzle, Spiral Galaxies [6].

2 Gadget Area Hardness Framework

Puzzles. The ***gadget area hardness framework*** applies to a general ***puzzle type*** (e.g., Tatamibari or Spiral Galaxies) that defines puzzle-specific mechanics. In general, a ***subpuzzle*** is defined by an embedded planar graph, whose finite faces are called ***cells***, together with an optional ***clue*** (e.g., number or symbol) in each cell. The puzzle type defines which subpuzzles are valid ***puzzles***, in particular, which clue types and planar graphs are permitted, as well as any additional **properties** guaranteed by a hardness reduction producing the puzzles.

We will use the unrestricted notion of subpuzzles to define gadgets. Define an ***area*** of a puzzle to be a connected set of cells. An ***instance*** of a subpuzzle in a puzzle is an area of

[1] While this gap may be caused by the puzzle being difficult to prove hard or simply overlooked (or both), we can confirm that it took us nearly six years to write this paper.

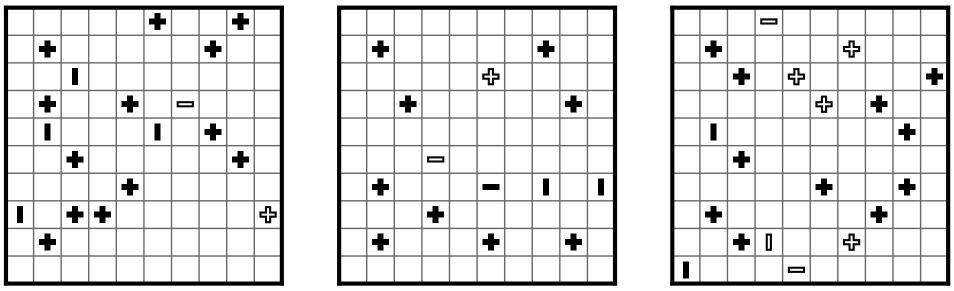


Figure 1 What do these Tatamibari puzzles spell when solved and the dark clues' rectangles are filled in? Figure 14 gives a solution.

the puzzle such that the restriction of the puzzle to that area (discarding all cells and clues outside the area) is exactly the subpuzzle.

Solutions. An ***area assignment*** (potential solution) for a (sub)puzzle is a mapping from clues to areas such that (1) the areas disjointly partition the cells of the (sub)puzzle, and (2) each area contains the cell with the corresponding clue. The puzzle type defines when an area assignment is an actual ***solution*** to a puzzle or a ***local solution*** to a subpuzzle.

Gadgets. A ***gadget*** is a subpuzzle plus a partition of its ***entire*** area (all of its cells) into one ***mandatory*** area and two or more ***optional*** areas, where all clues are in the mandatory area. A hardness reduction using this framework should compose puzzles from instances of gadgets that overlap only in optional areas, and provide a ***filling algorithm*** that defines which clues are in the areas exterior to all gadgets. Each gadget thus defines the entire set of clues of the puzzle within the gadget's (mandatory) area.

For a given gadget, a ***gadget area assignment*** is an area assignment for the subpuzzle that satisfies three additional properties:

1. the assigned areas cover the gadget's mandatory area;
2. every optional area is either fully covered or fully uncovered by assigned areas; and
3. every assigned area lies within the gadget's entire area.

A ***gadget solution*** is a gadget area assignment that is a local solution as defined by the puzzle type.

Profiles. A ***profile*** of a gadget is a subset of the gadget's entire area. A profile is ***proper*** if it satisfies two additional properties:

1. the profile contains the mandatory area of the gadget; and
2. every optional area of the gadget is either fully contained or disjoint from the profile.

Every gadget area assignment induces a proper profile, namely, the union of the assigned areas.

A profile is ***locally solvable*** if there is a gadget solution with that profile. A profile is ***locally impossible*** if, in any puzzle containing an instance of the gadget, there is no solution to the entire puzzle such that the union of the areas assigned to the clues of the gadget instance is that profile. These notions might not be negations of each other because of differences between local solutions of a subpuzzle and solutions of a puzzle.

Each gadget is characterized by a ***profile table*** (like a truth table) that lists all profiles that are locally solvable, and for each such profile, gives a gadget solution. A profile table is ***proper*** if it contains only proper profiles. A profile table is ***complete*** if every profile not in the table is locally impossible. A hardness reduction using this framework should prove that the profile table of each gadget is proper and complete, in particular, that any improper profile is locally impossible.

Given a puzzle containing some gadget instances, a ***profile assignment*** specifies a profile for each gadget such that the profiles are pairwise disjoint and the union of the profiles covers the union of the entire areas of the gadgets. In particular, such an assignment decides which overlapping optional areas are covered by which gadgets. A profile assignment is ***valid*** if every gadget is locally solvable with its assigned profile, i.e., every assigned profile is in the profile table of the corresponding gadget.

A hardness reduction using this framework should prove that every valid profile assignment can be extended to a solution of the entire puzzle by giving a ***composition algorithm*** for composing local solutions from the profile tables of the gadgets, possibly modifying these local solutions to be globally consistent, and extending these solutions to assign areas to clues from the filling algorithm (exterior to all gadgets).

3 Tatamibari is NP-hard

In this section, we prove Tatamibari NP-hard by a reduction from planar rectilinear monotone 3SAT. In Section 3.1 we briefly discuss a more constrained (but still NP-hard) variant of the classic 3SAT problem from which we will make our reduction; in Section 3.2, Section 3.3, and Section 3.4, we describe the gadgets (wires, variables, and clauses) from which we build the reduction; in Section 3.5, we discuss how the spaces between the gadgets are filled; and in Section 3.6 we use everything to show the main result.

3.1 Reduction Overview

We reduce from *planar rectilinear monotone 3SAT*, proved NP-hard in [9]. An instance of planar rectilinear monotone 3SAT comes with a planar rectilinear drawing of the clause-variable graph in which each variable is a horizontal segment on the x-axis and each clause is a horizontal segment above or below the axis, with rectilinear edges connecting variables to the clauses in which they appear. Each clause contains only positive or negative literals (i.e., is monotone); clauses containing positive (negative) literals appear above (below) the variables. We can always lengthen the variable and clause segments to remove bends in the edges, so we assume the edges are vertical line segments. We can further assume that each clause consists of exactly three (not necessarily distinct) literals: if a clause has $k < 3$ literals, we can just duplicate one of the clauses $3 - k$ times, which is easy to do while preserving the tri-legged rectilinear layout.

We create and arrange our gadgets directly following the drawing, possibly after scaling it up; see Figure 2. Edges between variables and clauses are represented by *wire gadgets* that communicate a truth value in the parity of their covering. For each variable, we create a *variable gadget*, which is essentially a block of wires forced to have the same value, and place it to fill the variable's line segment in the drawing. For each clause, we create a *clause gadget* with three wire connection points and place it to fill the clause's line segment. Negative clauses and wires representing negative literals are mirrored vertically. Both the variable and clause gadgets can telescope to any width to match the drawing; unused wires from the variable gadgets are terminated at a *terminator*. By our assumption that the edges are vertical segments, we do not need a turn gadget.

Covering a clause gadget without double-covering or committing a four-corner violation requires at least one of its attached wires to be covered with the satisfying parity (the true

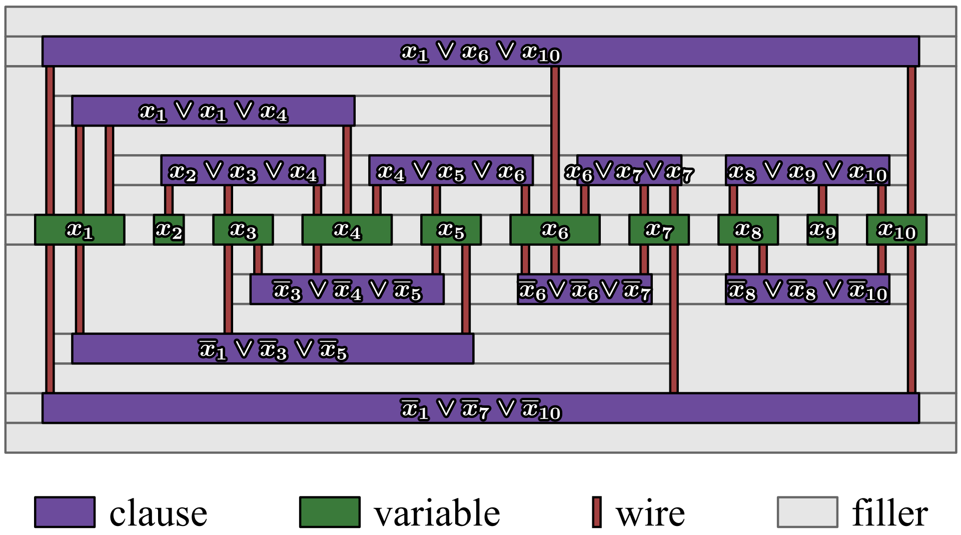

$x_1 \vee x_6 \vee x_{10}$
$x_1 \vee x_1 \vee x_4$
$x_2 \vee x_3 \vee x_4$
$x_4 \vee x_5 \vee x_6$
$x_6 \vee x_7 \vee x_7$
$x_8 \vee x_9 \vee x_{10}$
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
$\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_5$
$\overline{x}_6 \vee \overline{x}_6 \vee \overline{x}_7$
$\overline{x}_8 \vee \overline{x}_8 \vee \overline{x}_{10}$
$\overline{x}_1 \vee \overline{x}_3 \vee \overline{x}_5$
$\overline{x}_1 \vee \overline{x}_7 \vee \overline{x}_{10}$
clause
variable
wire
filler

Figure 2 The overall layout of the Tatamibari puzzles produced by our reduction follows the input planar rectilinear monotone 3SAT instance. Clause, variable, and wire gadgets are represented by purple, green, and red rectangles. Not drawn are terminator gadgets at the base of all unused copies of variables. Grey rectangles correspond to individual filler clues. Figure inspired by [9, Figure 2].

parity for positive clauses and the false parity for negative clauses).

To ensure clues in one gadget do not interfere with other gadgets, the wire gadget is surrounded on its left and right sides by sheathing of ◫ clue rectangles and the clause gadget is surrounded on three sides by a line of ⊞ clues forced to form 1×1 rectangles. Wire sheathing also ensures neighboring wires do not constrain each other, except in variable gadgets where the sheathing is deliberately punctured.

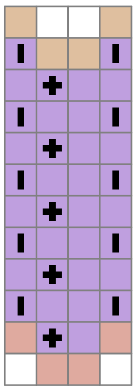
In our construction, gadgets will not overlap in their mandatory areas, so in the intended solutions, the mandatory area will be fully covered by rectangles satisfying the gadget's clues. Also in our construction, every optional area will belong to exactly two gadgets, and in the intended solutions, such an area will be covered by clues in exactly one of those gadgets.

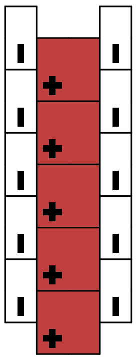
To apply the gadget area hardness framework, we define a ***local solution*** of a subpuzzle to be a disjoint set of rectangles satisfying the gadget's clues and the property that no four of these rectangles share a corner. (At the boundary of the subpuzzle, there is no constraint.) Our composition algorithm will combine these local solutions by staggering rectangles to avoid four-corner violations on the boundary of and exterior to gadgets. We will prove that valid profile assignments correspond one-to-one to satisfying truth assignments of the 3SAT instance.

We developed our gadgets using a Tatamibari solver based on the SMT solver Z3 [10]. The solver and machine-readable gadget diagrams are available [4]. Unfortunately, the solver can only verify the correctness of constant-size instances of the gadgets, but the variable and clause gadgets must telescope to arbitrary width. Thus we still need to give manual proofs of correctness.

3.2 Wire Gadgets and Terminators

The wire gadget, shown in Figure 3, consists of a column of ⊞ clues surrounded by ◫ clues which encodes a truth value in the parity of whether the squares are oriented with the ⊞


(a) An unsolved wire gadget. Mandatory area is purple and optional areas are brown.


(b) Wire communicating false

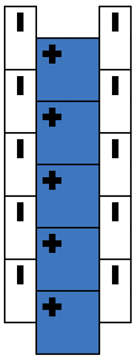
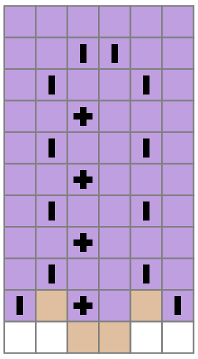
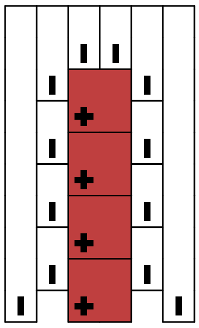

(c) Wire communicating true

Figure 3 Wire gadget and its profile table. The wire can be extended to arbitrary height by repeating rows. Note that between figures (b) and (c), the clues stay in the same place (and the rectangles shift to represent the different values of the wire).


(a) An unsolved terminator gadget. Mandatory area is purple and optional area is brown.


(b) Terminating a false wire

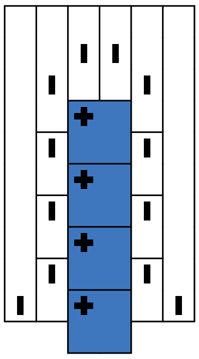

(c) Terminating a true wire

Figure 4 Terminator gadget and its profile table.

clues in their upper left or lower left corners. We will call this the *wire parity* or *wire value*. In this construction, only vertical wires are needed, and thus we do not give a turn gadget or horizontal wire. We call the column containing the ⊞ clues and the empty column next to it the *inner wire*. The inner wire is covered by columns of alternating ◫ clues, called the *(inner) sheathing*. In the overall reduction, ◫ clues in columns just outside the wire at its ends (in the variable gadget and either the clause or terminator gadget) add a further layer of sheathing (called the *outer sheathing*) outside the wire gadget, ensuring neighboring wires do not constrain each other.

The following lemmas will show that the ⊞ clues in the wire must be covered by 2×2 squares, the squares must all have the same parity, and the wire does not impart any significant constraints onto the surrounding region. These lemmas assume that no rectangle from a ◫ clue can reach the cells to the right of the top and bottom ⊞ clues in the wire, a property which we call ***safe placement***. We discharge this assumption in Section 3.5 by showing all wire gadgets produced by our reduction are safely placed.

▶ **Lemma 3.1.** *Each* ⊞ *in a safely placed wire covers a* 2×2 *square in the wire.*

Proof. There is no 3×3 square in the wire that contains a ⊞ clue but does not contain any other clue. Thus we cannot cover the ⊞ clue by squares larger than 2×2.

Now suppose we cover a ⊞ clue by a 1×1 square. Now the cells immediately above and below this clue must be covered. The ◫ clues must be taller than they are long, so they cannot cover these cells. Thus we must cover them by squares containing the ⊞ clues above and below. This leaves the cell directly to the right of the 1×1 uncovered. It is easy to see that this cannot be covered by the nearby ⊞ clues or ◫ clues. The cells next to the top and bottom clues cannot be covered by a ◫ clue from outside the gadget by our assumption that the wire is safely placed.

The only remaining possibility is a ⊟ clue from outside the gadget extending into the wire gadget. Such a rectangle cannot extend entirely through the wire because the ◫ clues in the sheathing and the ⊞ clues inside the wire are in alternating rows. If the external horizontal rectangle enters the wire from the right and covers a cell next to the ⊞ clue, that ⊞ clue is forced to be a 1×1 rectangle and the cell above it must be covered by the next ⊞ clue above. This results in a four-corner violation involving the two ⊞ clues and the left sheathing except when the ⊞ clue is at the bottom of the wire. In that case, the external horizontal rectangle blocks the bottom-right sheathing clue, making it 1×1 and unsatisfied. ◀

▶ **Corollary 3.2.** *Satisfied safely placed wires must have all of their 2×2 squares with the ⊞ clues in the lower left corner or all in the upper left corner.*

Proof. By Lemma 3.1 all ⊞ clues must be covered by 2×2 squares. To change whether the ⊞ clues are in the lower left or upper left, we will end up leaving a row of two cells between clues blank. By the same arguments in Lemma 3.1 these cannot be covered by the nearby ⊞ clues or ◫ clues. ◀

▶ **Lemma 3.3.** *The ◫ clues making up the inner sheathing of satisfied safely placed wires must be covered by 1×2 rectangles of opposite parity to the wire's squares.*

Proof. By Corollary 3.2 the wire has one of two parities of squares. If a vertical rectangle ends with the same y coordinate as an adjacent square, then we will have three right angles at a single corner, forcing a four-corner violation or uncovered cell. Because the squares are 2×2, a vertical rectangle of odd height guarantees one of the ends will share a y-coordinate with one of the squares. The ◫ clues occur every other cell, so the vertical rectangles cannot be of length greater than 3. This forces them to be of length 2 and staggered with respect to the squares. ◀

▶ **Theorem 3.4.** *The safely placed wire gadget's profile table is proper and complete.*

Proof. By Lemma 3.1, each optional area must be fully covered or fully uncovered by the neighboring ⊞ clue, so the profile table is proper. Corollary 3.2 fixes the ⊞ clue parity and Lemma 3.3 fixes the sheathing parity, so all other profiles are locally impossible, so the profile table is complete. ◀

We also have a terminator gadget to terminate unused wires regardless of their parity. The terminator gadget is shown in Figure 4.

▶ **Lemma 3.5.** *The terminator does not constrain the wire parity.*

Proof. Figures 4b and 4c show solutions of the terminator with both parities. The same arguments about wire correctness show this gadget does not allow any additional wire solutions nor constrain other gadgets. ◀

▶ **Theorem 3.6.** *The terminator gadget's profile table is proper and complete.*

Proof. The profile table in Figure 4 contains only proper profiles, so the profile table is proper. By the same arguments in Lemma 3.1, the two local solutions shown are the only way to cover the wire part of the gadget. A horizontal rectangle from a ⊟ outside the gadget could cover part of the top row of the gadget (or the entire top row when terminating a true wire) while leaving the clues in the gadget satisfied and covering the remaining area. We prevent this through the global layout: all clause gadgets (the only gadget containing ⊟ clues) appear strictly above (for positive clauses) or strictly below (for negative clauses) all terminator gadgets, so it is not possible for any ⊟ rectangles to cover area in the clause gadget. Thus all other profiles are locally impossible, so the profile table is complete. ◀

3.3 Variable Gadgets

The variable gadget is essentially a series of wires placed next to each other with devices we call ***couplers*** in between. Each coupler acts essentially as an "equality" constraint between neighboring wires, thus forcing all the wires connected via a series of couplers to represent the same variable; this collection of wires then forms the ***variable gadget*** of the reduction.

Each coupler takes two columns, and consists of (i) a ⊞ clue which interacts with the inner sheathing of the wires to force equality, and (ii) eight ◫ clues (two above and two below the ⊞ clue on each column), which prevent the inner sheathings of the neighboring wires from constraining each other (except through the ⊞ clue itself). See Figure 7 for an example with two wires; additional wires can be added to either side of variable by using more couplers (see Figure 6).

First, notice that both wires are constrained to have their squares in one of two parities by the inner sheathing, as in Corollary 3.2. It is also important that wires do not constrain each other outside the couplers, either directly (if they happen to be adjacent) or indirectly (through the space in between); we address this in Section 3.5.

Now we have to show that two wires separated by the coupler must be in the same configuration. This happens because the wire parity forces the parity of the inner sheathing, which forces the parity of the coupler, which then forces the partiy of the inner sheathing and the wire parity of the next wire over.

▶ **Lemma 3.7.** *The coupler has only two valid coverings of its* ⊞ *clue.*

Proof. The location of the eight ◫ clues around the ⊞ clue ensure that it cannot be larger than 2×2. By Corollary 3.2 we know that the wire gadgets next to the coupler must have their inner sheathing as 2×1 rectangles in either the up or down position. If the ⊞ clue is covered by a 1×1 it will create a four-corner violation with the inner sheathing. Thus it must be one of the two possible positions for a 2×2 square. If both inner sheathings have the same parity, as in Figure 7 then the constraints can be locally satisfied. ◀

▶ **Lemma 3.8.** *All wires in a variable gadget must have the same value (i.e. upwards branches must have the same orientation).*

Proof. We know the coupler has at most two ways to satisfy its constraints, corresponding to a 2×2 square in either the up or down position. Notice that the inner sheathing of both wires must be of different parity from the square or they will cause a four-corner violation. Thus the inner sheathing must have the same parity, ensuring that the wires themselves must have the same parity. If multiple wires are all connected by couplers, then they will all be forced to have the same parity by the same local argument. ◀

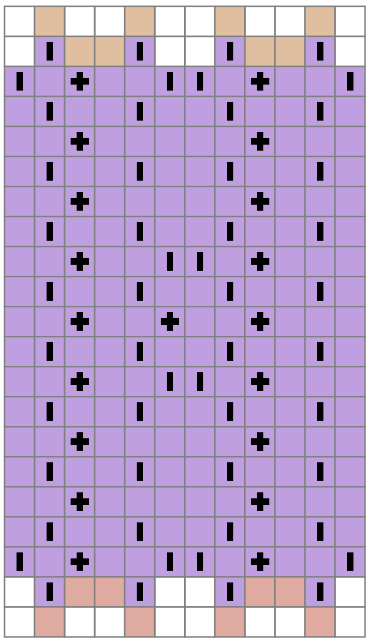


Figure 5 The variable gadget. Mandatory area is purple and optional areas are brown.

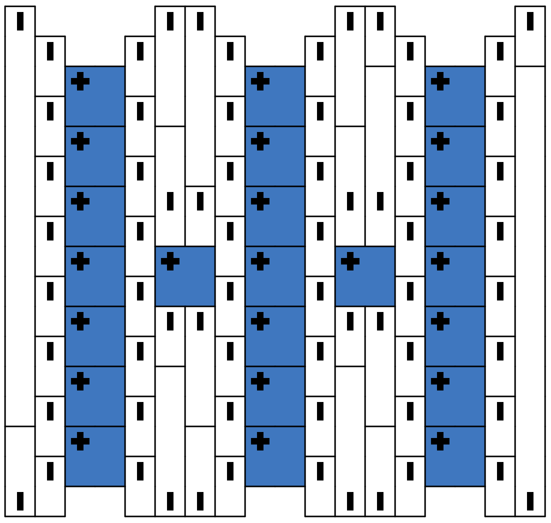


Figure 6 A variable gadget widened to provide three wires, shown here set to true.

▶ **Lemma 3.9.** *The variable gadget is locally solvable with a given profile if and only if the profile satisfies (i) all upwards branches have the same orientation, (ii) all downwards branches have the same orientation, and (iii) upwards and downwards branches have opposite orientations from each other.*

Proof. The "only if" direction follows from Lemma 3.8 and Corollary 3.2 (each wire individually must have opposite orientation for upwards and downwards branches due to the couplers, and all wires in the gadget must have the same upward orientation).

The "if" direction follows from Lemma 3.5, the individual solvability of each wire and terminator in both orientations (as shown in Figure 3b, Figure 3c, Figure 4b, and Figure 4c), and the solvability of the couplers given that adjacent wires have the same orientation (Figure 7). Neighboring wires (within the variable gadget) do not conflict with each other (outside of the coupler) because of the "outer sheathing" columns separating them; the meeting points of the two clues in each "outer sheathing" column can be adjusted to avoid four-corner violations with each other, as well as avoiding four-corner violations with the neighboring "inner sheathing". ◀

Note that this lemma is what we want from a variable gadget: it is locally solvable if and only if its profile corresponds to a specific value for the variable it represents.

3.4 Clause Gadgets

The clause gadget, shown in Figure 8, interfaces with three wire gadgets representing the three literals of this clause. In the upper-left of the variable gadget is an internal wire, which we call the ***clause verification wire***. The only way to cover the top two cells of that wire is using the wire's top ✚ clue. This is only possible when at least one of the wires is true,

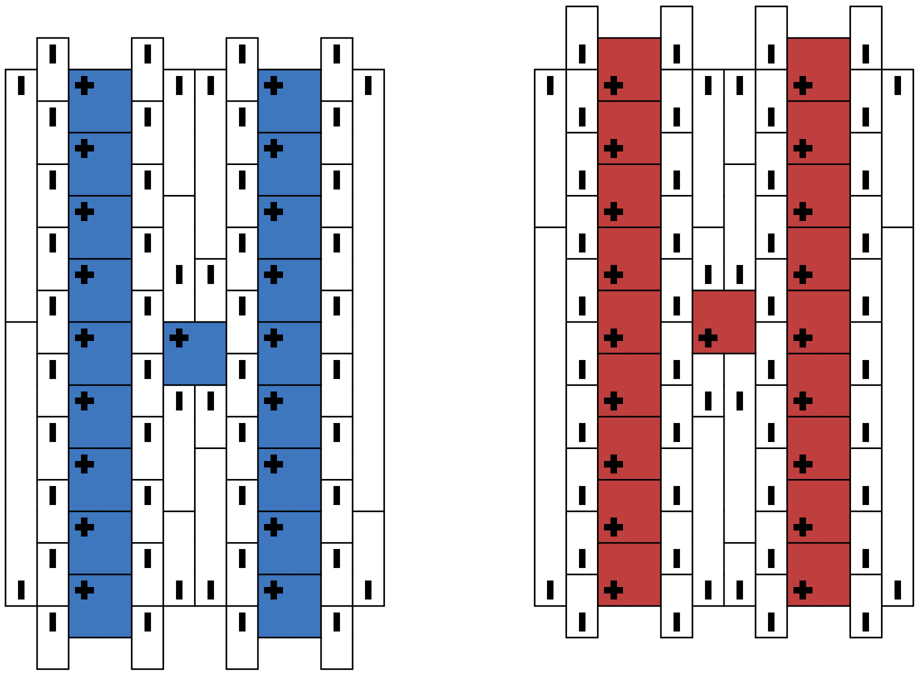


Figure 7 The variable gadget's profile table. Left: variable set to true. Right: variable set to false.

allowing a ***variable enforcement line*** (drawn in figures as a purple horizontal bar) to provide a parity shift to the clause verification wire. Otherwise, either those top two cells cannot be covered, or some other cell in the clause will not be covered, or there will be a four-corner violation.

The mandatory areas of the clause include all clues and cells shown in Figure 11 and optional areas consisting of the row of cells at the bottom of the gadget, specifically the set of cells under the ▮ clue lines at the bottom of the gadget.

Each of the three wires in this gadget has two intended solutions: true or false. In Figure 11, the wire is blue if it represents true and red if it represents false. The leftmost wire behaves somewhat differently from the others because it is closest to the clause verification wire.

Importantly, the clause gadget can be expanded horizontally such that the variable wires can be spaced an arbitrary amount beyond the width of the base gadget shown in Figure 11. The columns between the literal wires in the clause gadget can be expanded an arbitrary number of columns. Such an example expansion is shown in Figure 9. In this example, the columns have been expanded such that the entire gadget is wider by 4 columns and the number of columns between each literal in the gadget has been expanded by 2 columns.

▶ **Lemma 3.10.** *If any wire is in the* false *configuration, then the variable enforcement line corresponding to the wire will not be able to go across the gadget.*

Proof. If a wire is in the false configuration, then there exists at two cells on the top of the wire that need to be covered. These two cells can be covered in two different ways. We first prove this lemma for the leftmost wire and then prove the lemma for the other wires since the leftmost wire is different from the others. In this case, the only way to cover the two cells is with a 2×2 square (see Figure 10), blocking the variable enforcement line from crossing

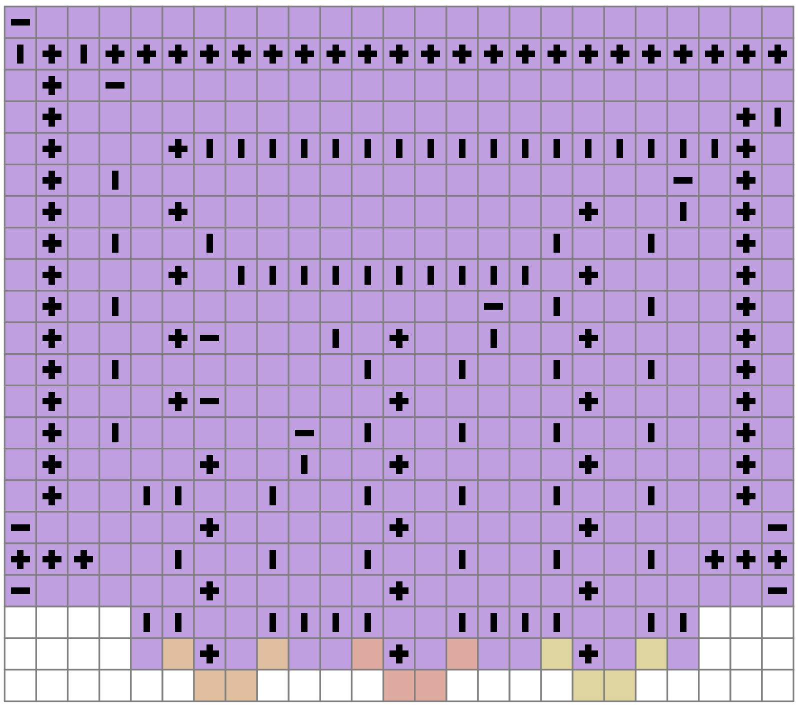


Figure 8 An unsolved clause gadget. Mandatory area is purple and optional areas are brown.

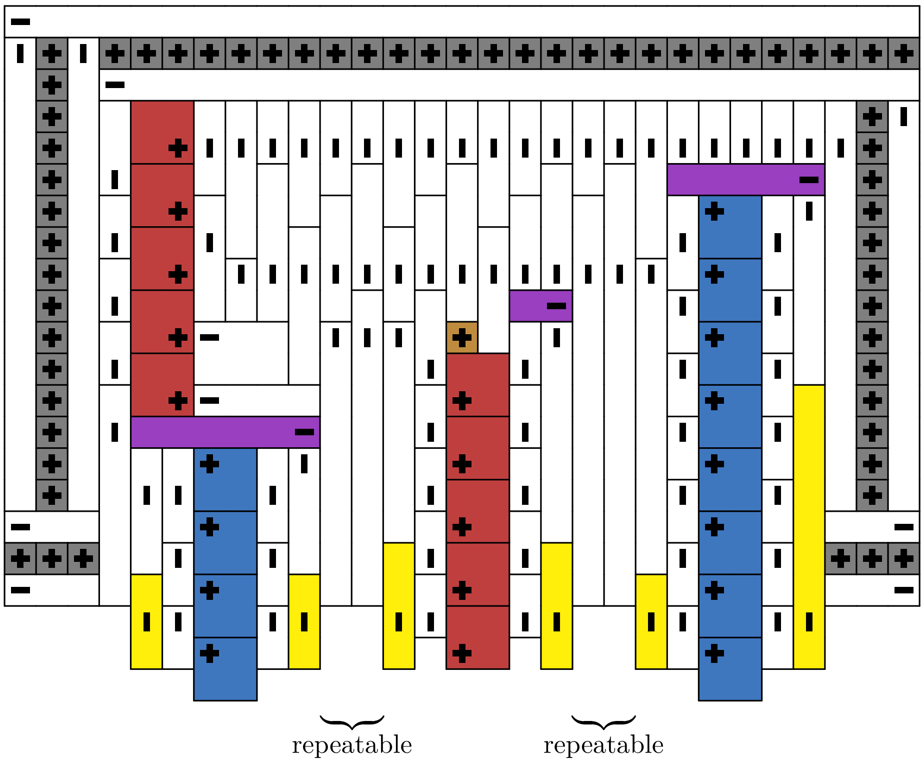


Figure 9 Example where the columns in between literal wires in the clause gadget have been expanded. The columns which are able to be repeated an arbitrarily number of times have been labeled as "repeatable" in the figure since they can be repeated an arbitrarily number of times to make the clause an arbitrary width.

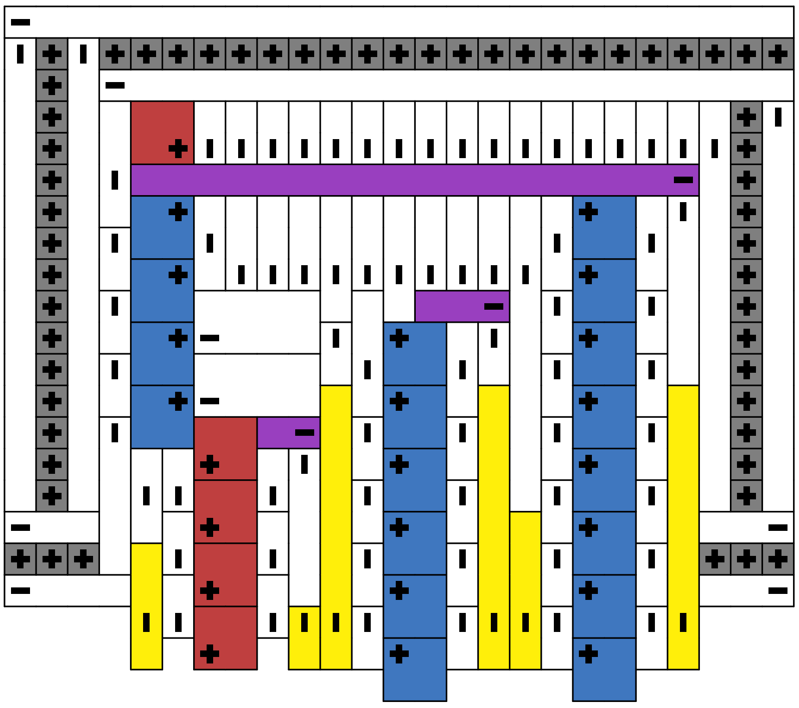


Figure 10 When the leftmost wire is set in the false configuration, the only way to cover the top two cells of the wire is with a 2×2 square that blocks the variable enforcement line.

the top of the wire.

For the other two wires, the top two cells can be covered in only two ways. Either a 1×1 square covers one of the two cells and a vertical line from the top covers the other cell or vice versa (see Figure 11).

No other configurations are available that does not violate the four-corner constraint. Thus, this configuration prevents the corresponding variable enforcement lines from going across the gadget. ◀

▶ **Corollary 3.11.** *When all wires in the gadget are false, the puzzle does not have a solution.*

Proof. By Lemma 3.10, no variable enforcement line can go across the gadget if all wires are false. In order to solve the puzzle presented by the gadget, the top two cells of the clause verification line must be covered. These two cells cannot be covered by the horizontal line on top of them nor can they be covered by the vertical lines beside them. Thus, they must be covered by the 2×2 square formed in the clause verification line. However such a square will either leave a cell in the middle of the clause verification line uncovered or will leave the bottom two cells of the line uncovered. In this case, no configurations exist in covering these bottom two cells without violating the four-corner rule. See Figure 11a. Thus, the gadget is unsatisfiable if all wires into the gadget are false. ◀

▶ **Lemma 3.12.** *If at least one of the wires entering the clause gadget is in the true configuration, then the clause gadget is locally solvable.*

Proof. In any wire is in the true configuration, then the variable enforcement line corresponding to the gadget will be able to go across the gadget. For the leftmost wire, the clause verification line will be in the configuration that ensures that all cells that need to be covered by the line are covered. Otherwise, the variable enforcement line will be able to cause the clause verification line to cover all the necessary cells. See Figures 11b to 11h. ◀

Using the above lemmas, we are able to prove the following properties of the profile table of the clause gadget.

▶ **Corollary 3.13.** *The profile table of the clause gadget is proper.*

▶ **Lemma 3.14.** *The profile table of the clause gadget is complete.*

Proof. The clause gadget's profile table contains all profiles shown in Figure 11 except for the all-false configuration shown in Figure 11a. By Corollary 3.11, the all-false configuration is not locally solvable. It remains to show the all-false configuration is locally impossible.

To do this, we show that no solution to a clue outside of this profile is able to solve any part of the all-false clause profile–essentially that the clause gadget is fully isolated from the rest of the puzzle. By design, no clue above, to the left of, or to the right of the clause can cover any of the cells that are left uncovered by the literals, because the row and columns of single-cell squares blocks any rectangles from reaching the uncovered cells.

We now prove that no clues from the bottom of the gadget can help cover any of these cells. Such clues can only potentially cover the optional areas at the bottom of the gadget. We show that such clues cannot cover parts of the literal gadgets. By Lemma 3.7, there are only two possible configurations of the variable gadgets; thus, no other outside fillers can cover any cells in the incoming wires. Hence, no clues adjacent to the bottom of the gadget can help cover any part of the incoming wires.

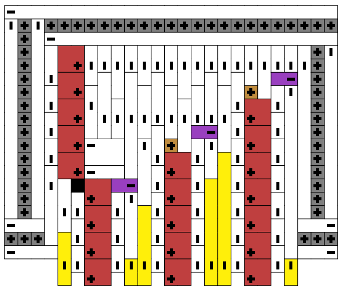
Thus the all-false profile is locally impossible, so the profile table is complete. ◀

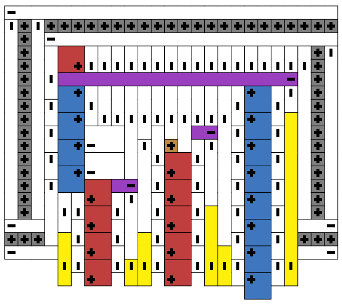
3.5 Layout, Sheathing, and Filler

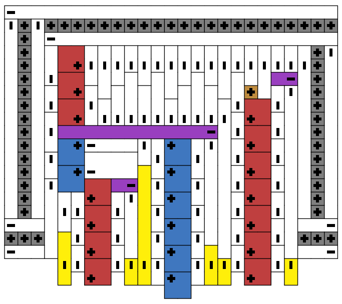
In order to build the full Tatamibari instance corresponding to a planar rectilinear monotone 3SAT instance, we lay out the gadgets as shown in Figure 2: variable gadgets are positioned on a central line, while positive and negative clauses are positioned above and below respectively at heights corresponding with how many layers of clauses are nested below them, with wires running vertically from variables to clauses (both variable and clause gadgets can be extended arbitrarily far horizontally). Variable and clause gadgets have rectangular profiles (except for where the wires "plug in" to them). Variables and clauses have a uniform height, and for any two variable or clause gadgets, they are placed on exactly the same set of rows or they share no rows.

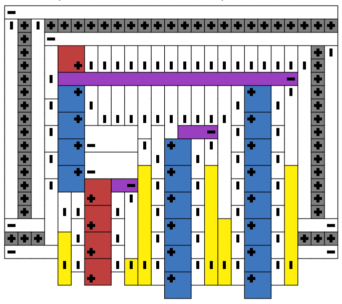
All wire gadgets in the puzzle produced by our reduction are safely placed; that is, no rectangle from a $\boxed{|}$ clue can reach the cells to the right of the top and bottom $\boxed{+}$ clues in the wire. The only $\boxed{|}$ clues in those columns are in clause gadgets. The row of single-cell squares at the top of the clause gadget blocks any rectangles from extending upwards out of the clause gadget. If a rectangle from a $\boxed{|}$ clue in those columns of the clause gadget extends downward past the first $\boxed{+}$ clue in the column to its left, the cell below that $\boxed{+}$ clue cannot be covered by any clue, so rectangles cannot extend downward out of the clause gadget in those columns. Thus $\boxed{|}$ clues from clause gadgets cannot interact with wire gadgets, so the wires are safely placed.

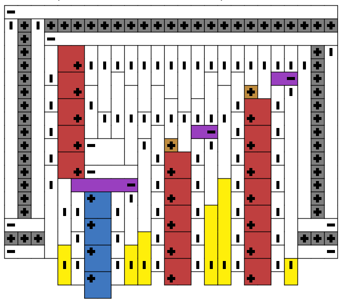
Because we want the solvability of the Tatamibari instance to depend only on solving the gadgets, we need to add ***filler*** clues that are always able to cover the areas outside the gadgets.

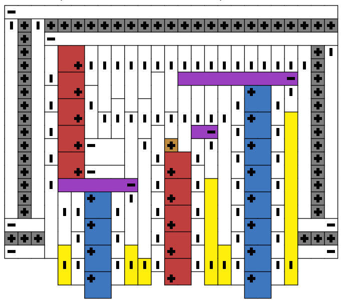

(a) The (false, false, false) configuration.

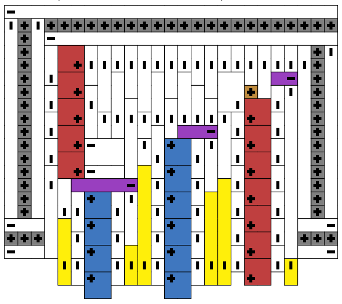

(b) The (false, false, true) configuration.


(c) The (false, true, false) configuration.


(d) The (false, true, true) configuration.


(e) The (true, false, false) configuration.


(f) The (true, false, true) configuration.


(g) The (true, true, false) configuration.

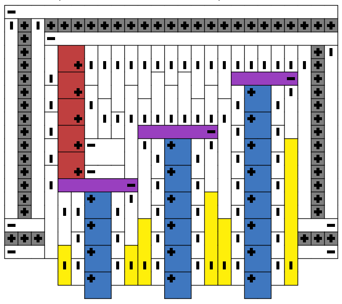

(h) The (true, true, true) configuration.

Figure 11 The clause gadget. All configurations shown here except the all-false configuration in Figure 11a are in the clause gadget profile table. Clues highlighted in yellow also function as the "outer sheathing" protecting the wires closest to them (see Section 3.5). For the false wires, the only configuration that guarantees the two cells at the top are covered are the cases where one 1×1 square covers one (shown in brown) and a long rectangle extending from the top covers the other.

First, we set aside any cells horizontally adjacent to a wire gadget. These cells will be covered by the outer sheathing clues described in described in Section 3.2 and Section 3.3 and highlighted in yellow in Figure 9 and Figure 11. In the global solution, the areas assigned to the outer sheathing clues thus extend vertically outside their gadget. For the purposes of the filler algorithm, we consider the cells covered by the outer sheathing to be part of the wire gadget.

Each filler clue corresponds to a rectangular area of space between gadgets, formed by breaking each row into maximal horizontally contiguous strips between (and bordered by) the gadgets, then joining vertically contiguous strips into a single rectangle if they have the same width. The filler algorithm places a single clue in each of these rectangles (◫, ⊟, or ⊞ depending on the rectangle's aspect ratio), placed arbitrarily inside (say, in the upper-right corner). See Figure 2 for an example. While it may be possible for the solver to use these clues differently than shown here, we only need to prove that if the solver does assign each rectangular area to its associated clue, it will cover the area.

The only potential problem lies in the possibility of a four-corner violation involving a filler rectangle. This can only happen where either (i) a corner of a filler rectangle meets a gadget and a wire coming from that gadget, or (ii) where two corners of filler rectangles meet along the edge of a gadget. If a corner of a filler section meets an edge of another filler section or the edge of the board there cannot be a four-corner violation.

Remark: There is a potential third problem case, where two wires are directly adjacent with only the outer sheathing (2 columns) between them (see Figure 8, which has this property). This can be dealt with in either of two ways: ensuring that no wires are directly adjacent to each other by stretching the instance horizontally, or noting that the meeting points of the outer sheathing of the two adjacent wires can be adjusted to not produce a four-corner violation between them.

▶ **Proposition 3.15.** *If the gadgets can all be satisfied, the filler clues can also be satisfied.*

Proof. Each filler clue will be satisfied by a rectangle covering its entire associated area; the cells horizontally adjacent to wires will be filled by two width-1 vertical rectangles from the outer sheathing clues, one coming from the clause gadget above and the other coming from the variable gadget below. The meeting point between the two outer sheathing rectangles can be adjusted as needed to avoid a four-corner violation. As mentioned, we have only two problem cases: (i) a corner of the filler rectangle meets a gadget and protruding wire; and (ii) corners of two sections meet on the side of a wire. Because both cases involve the side of a wire, we can avoid violations in either case by appropriately adjusting the meeting point of the sheathing clues.

(i) To avoid having a corner where the corner of the filler section meets the wire and gadget, the meeting point of the two sheathing clues can be placed on the edge (not corner) of the filler section, thus avoiding a four-corner violation since the corner of the filler section meets the edge of one of the sheathing rectangles.

(ii) As long as the meeting point of the two sheathing rectangles of the wire is not at the point where the two filler sections meet, there is no four-corner violation. The meeting point can trivially be placed on the side of a filler section (while still respecting the parity of the wire as expressed by the inner sheathing).

Therefore, since the sheathing can always be adjusted to accommodate filler rectangles, the satisfiability of the Tatamibari instance depends only on the gadgets. ◀

3.6 Finale

Now we can show that Tatamibari is NP-hard. Let f be the reduction, which takes an instance Φ of planar rectilinear monotone 3SAT and returns a Tatamibari instance $f(\Phi)$; we want to show:

▶ **Proposition 3.16.** *If Φ has n variables and m clauses, then $f(\Phi)$ has size polynomial in $n + m$, and can be computed in time polynomial in $n + m$.*

Proof. Our construction expands the given planar rectilinear monotone 3SAT instance by a constant factor. Therefore it suffices to prove that planar rectilinear monotone 3SAT is strongly NP-hard when given the coordinates of the rectilinear drawing. Indeed, the height of the drawing is $O(m)$ and the width of the drawing is $O(e)$ if the graph has e edges, which is $O(m + n)$ by planarity. ◀

▶ **Proposition 3.17.** *If Φ has a solution, then $f(\Phi)$ also has a solution.*

Proof. We begin by taking the solution to Φ and setting the variable gadgets' profiles according to those values; by Lemma 3.9, they will all be locally solvable. By Lemma 3.12, since each clause gadget is connected to wires representing variables which satisfy the clause, there must be a solution to the clause gadget. Furthermore, by Proposition 3.15, if the gadgets are satisfiable then the rest of the space can be filled without contradiction, producing a solution to $f(\Phi)$. ◀

▶ **Proposition 3.18.** *If Φ has no solution, then $f(\Phi)$ also has no solution.*

Proof. We prove the equivalent statement that if $f(\Phi)$ has a solution, then Φ must also have a solution.

First, we prove that any solution to $f(\Phi)$ must correspond to some setting of the variables $x_1, \dots, x_n$ of Φ. This is a consequence of Lemma 3.8, which shows that all wires in a single variable gadget must carry the same value, which is then taken as the setting for that variable.

Next, we have to show that these settings of the variables x_i are a solution of Φ; to do this, note that by Corollary 3.2 each wire ending in a clause gadget must carry its value into this clause gadget; and by Corollary 3.11 and Lemma 3.12 there is a solution to the clause gadget if and only if the wires represent values which satisfy the clause.

Thus, the values of the variable gadgets must be a solution to Φ. ◀

The above three propositions imply our desired result:

▶ **Theorem 3.19.** *Tatamibari is (strongly) NP-hard.*

Because a given Tatamibari solution can be trivially checked in polynomial time, this theorem implies that Tatamibari is NP-complete.

4 Font

Figure 12 shows a series of twenty-six 10×10 Tatamibari puzzles that we designed, whose unique solutions shown in Figure 13 reveal each letter A–Z. To represent a bitmap image in the solution of a Tatamibari puzzle, we introduce two colors for clues, light and dark, and similarly shade the regions corresponding to each clue. As shown in Figure 13, the letter is drawn by the dark regions from dark clues. These puzzles were designed by hand, using our SAT-based solver [4] to iterate until we obtained unique solutions. The font is also available online.[2]

[2] http://erikdemaine.org/fonts/tatamibari/

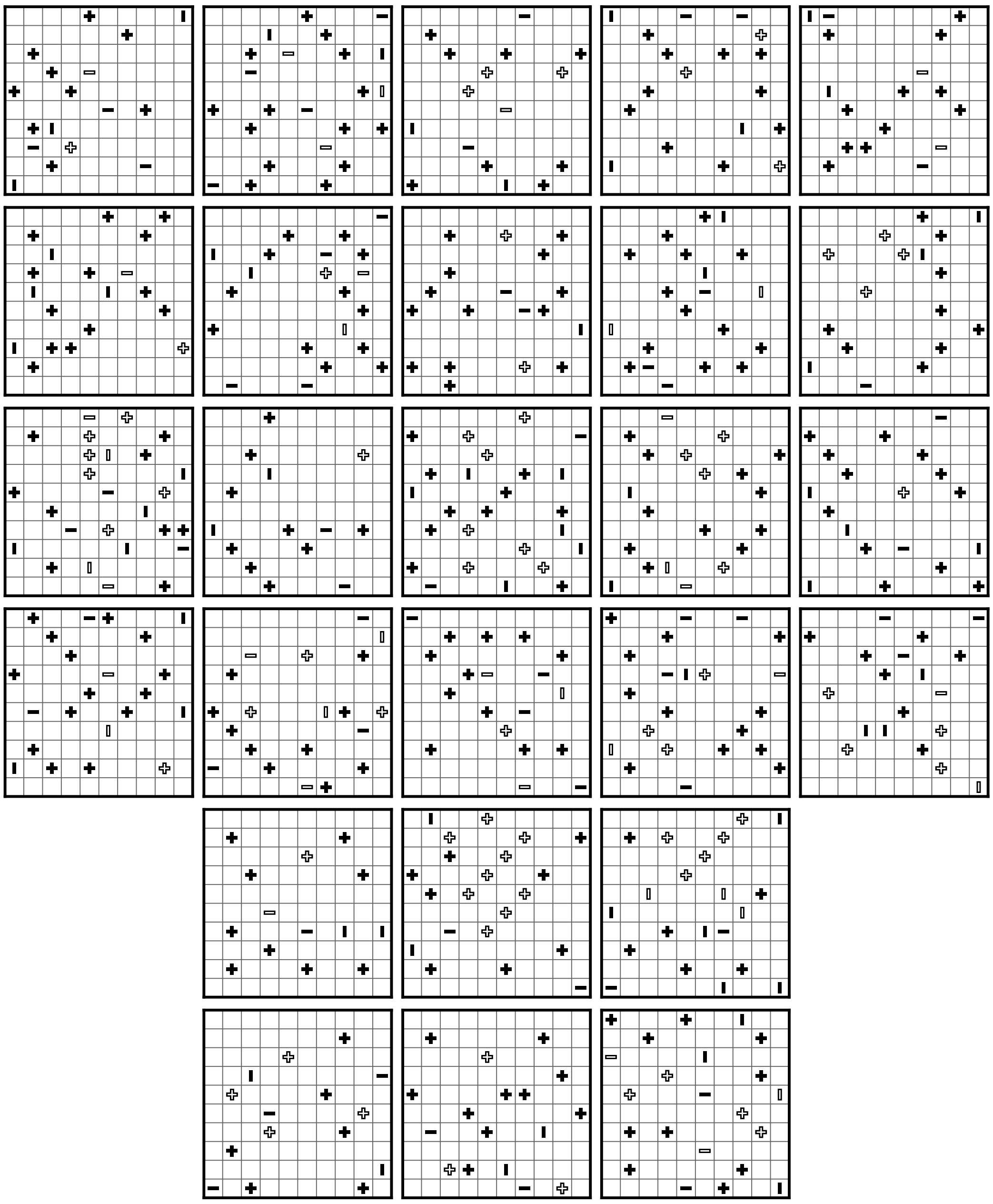


Figure 12 Puzzle font: each puzzle has a unique solution whose regions for dark clues (shown in Figure 13) form the shape of a letter.

5 Open Problems

There remain interesting open questions regarding the computational complexity of Tatamibari. When designing puzzles, it is often desired to have a single unique solution. We suspect that Tatamibari is ASP-hard (NP-hard to determine whether it has another solution, given a solution), and that counting the number of solutions is #P-hard. However, our reduction is far from parsimonious. Some rework of the gadgets, and a unique filler between gadgets, would be required to preserve the number of solutions.

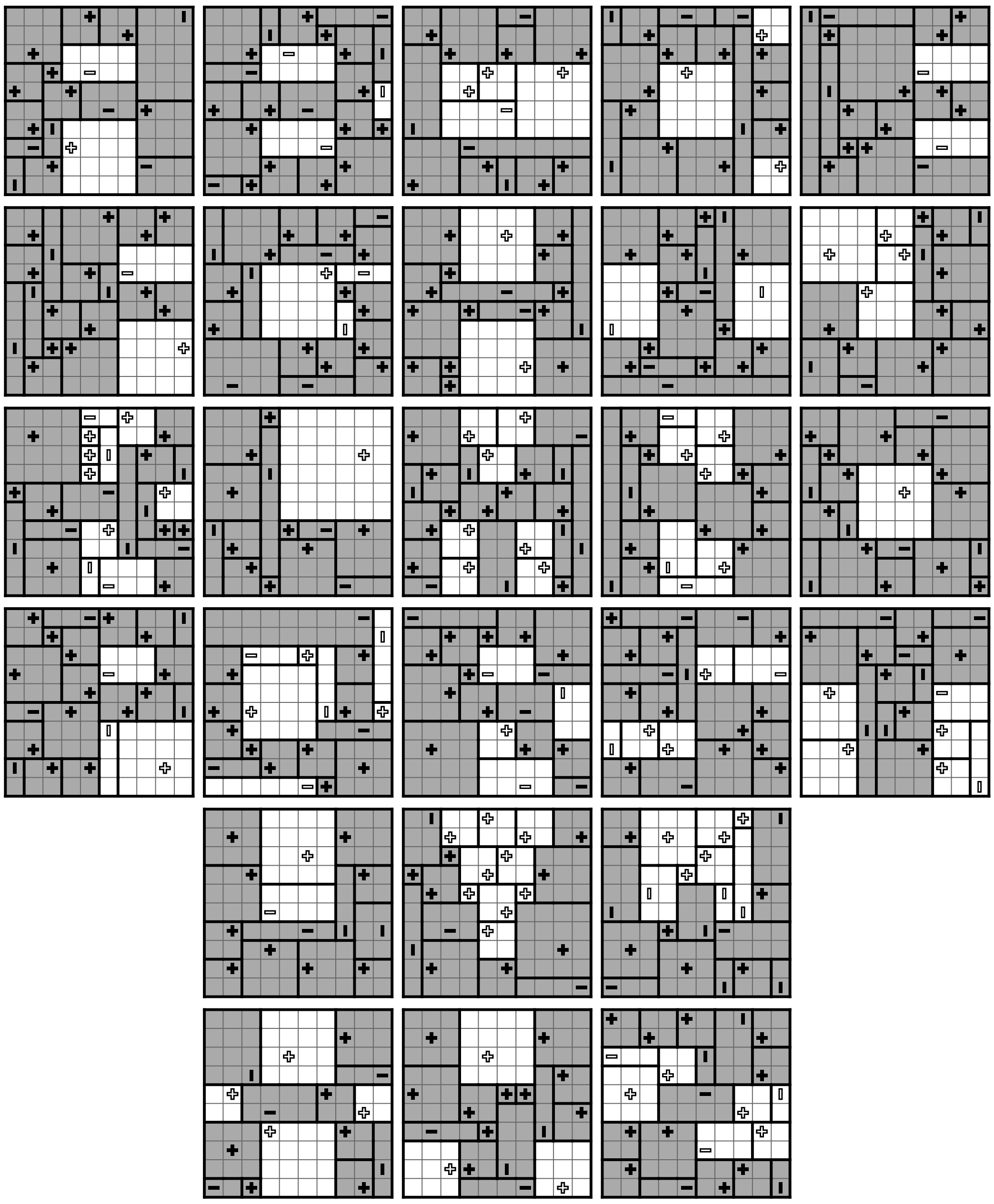


Figure 13 Solved font: unique solutions to the puzzles in Figure 12.

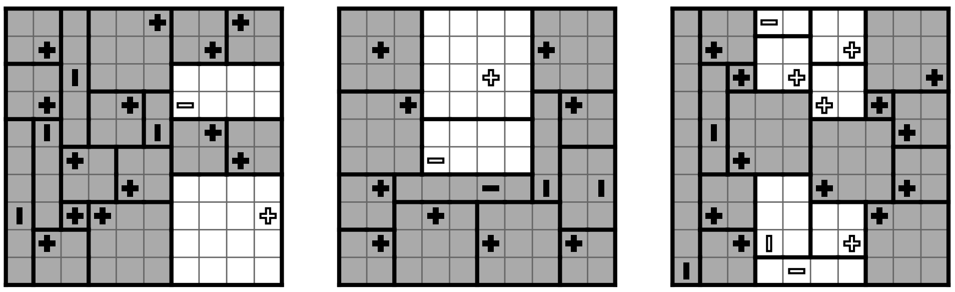
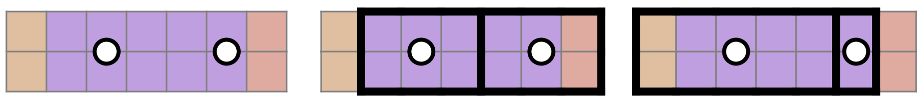


Figure 14 Solution to Figure 1.



(a) Unsolved wire gadget

(b) Wire carrying true signal: 3×2 rectangles

(c) Wire carrying false signal: alternating 1×2 and 5×2 rectangles

Figure 15 Spiral Galaxies wire and its profile table (true and false solutions)

We could ask about restrictions or natural variations of Tatamibari. For example, we are curious whether a Tatamibari puzzle with only ⊞ clues, or only ▯ clues, remains hard. We have also wondered about the version of the puzzle without the four-corner constraint. Although initially we thought of the four-corner constraint as a nuisance to be overcome in our reduction, our final proof uses it extensively and centrally.

A Example: Spiral Galaxies

As an example of the gadget area hardness framework, we show how the NP-hardness proof for Spiral Galaxies from [15] can be described using the framework. A Spiral Galaxies puzzle is a rectangular grid with clues in some grid cells or on some grid lines. The goal is to partition the puzzle into areas with a single clue per area such that the area is rotationally symmetric about the clue.

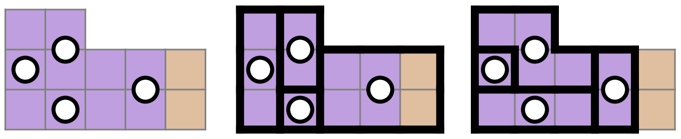
We reduce from planar[3] Boolean circuit satisfiability. We have a wire gadget, a variable gadget, NOT and AND gadgets, a fanout (wire duplicator) gadget, and a vertical shift gadget. We lay out these gadgets to overlap in their optional areas (only), and communicate a truth value in whether the optional area is covered or not.

Wire. The wire gadget consists of repeating pairs of clues three grid units apart. There are two gadget solutions, shown in Figure 15: repeating 3×2 rectangles, in which case the wire covers the right optional area, and alternating 1×2 and 5×2 rectangles, in which case the wire covers the left optional area. The wire carries a true signal when it covers the right optional area and false when it covers the left optional area. The wire gadget can be extended to arbitrary length in units of two clues. (The proof in [15] does not explicitly state this parity requirement, but the gate gadgets assume the true signal protrudes into the gadget to cover the optional input area and the false signal does not.)

Boolean circuit satisfiability requires the circuit produce a true output. We can force a wire to be true simply by terminating it. Because the wire has height two, any filler clues to the right of the wire cannot cover area in the wire gadget, so the wire must end in a 3×2 rectangle to cover the right optional area, forcing the rest of the wire to also carry a true signal.

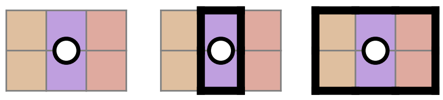
Variable. The variable gadget is shown in Figure 16. There are two gadget solutions, one leaving the optional area uncovered (so the adjacent wire is set to true) and the other covering it (so the adjacent wire is set to false). Choosing one solution or the other corresponds to assigning true or false to the variable.

[3] Friedman's proof [15] provides a crossover gadget, but it is not necessary because AND and NOT build a crossover [16].



(a) Unsolved variable gadget **(b)** Variable set to true **(c)** Variable set to false

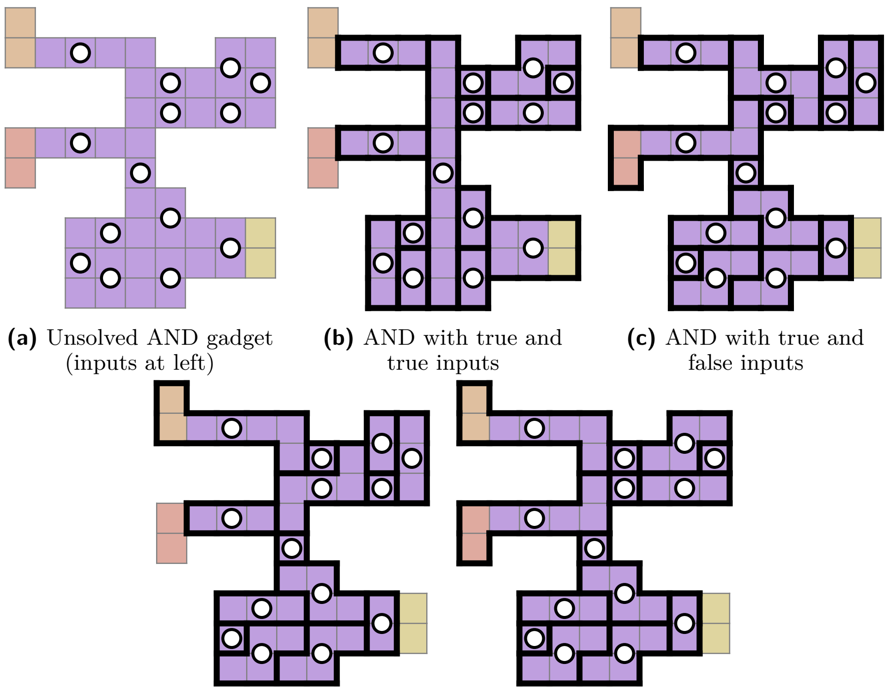
Figure 16 Spiral Galaxies variable and its profile table (true and false solutions)



(a) Unsolved NOT gadget **(b)** NOT gadget converting true to false **(c)** NOT gadget converting false to true

Figure 17 Spiral Galaxies NOT gadget and its profile table

NOT. The NOT gadget is shown in Figure 17. If the left optional area is covered by the input wire (carrying a true signal), the clue in the NOT gadget must cover a 1×2 rectangle, so the right optional area must be covered by the output wire carrying a false signal. If the left optional area is uncovered (when the input wire is false), the clue in the NOT gadget covers both optional areas, so the output wire must carry a true signal. Thus the NOT gadget inverts the wire's signal.



(a) Unsolved AND gadget (inputs at left) **(b)** AND with true and true inputs **(c)** AND with true and false inputs

(d) AND with false and true inputs **(e)** AND with false and false inputs

Figure 18 Spiral Galaxies AND gadget and its profile table

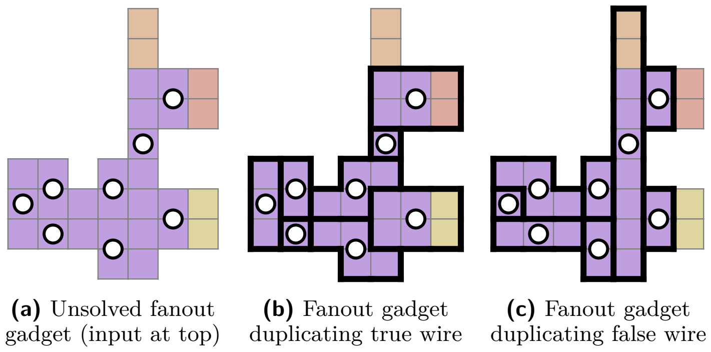

(a) Unsolved fanout gadget (input at top) **(b)** Fanout gadget duplicating true wire **(c)** Fanout gadget duplicating false wire

Figure 19 Spiral Galaxies fanout gadget and its profile table

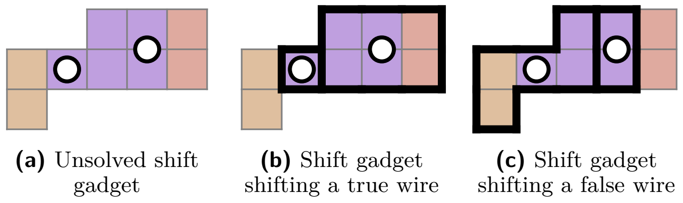

(a) Unsolved shift gadget **(b)** Shift gadget shifting a true wire **(c)** Shift gadget shifting a false wire

Figure 20 Spiral Galaxies upward shift gadget and its profile table; the downward shift gadget is this gadget flipped vertically

AND. The AND gadget is shown in Figure 18. When both inputs are true, both of the left optional areas are covered by the wire, so the clues to the right of the optional area are rectangles and the clue at the center of the gadget is a long vertical rectangle, allowing the right optional area to be covered, propagating a true signal from the gadget. When either or both of the inputs are false, one or both of the left optional areas must be covered by the clue(s) to the right of the areas, blocking the central clue from covering a vertical rectangle, preventing the right optional area from being covered, thus propagating a false signal from the gadget.

Fanout. Like the AND gadget, the fanout gadget (Figure 19) is also based around forming or not forming a vertical rectangle. The upper optional area is the input. When it is covered by the input wire (a true signal), the central clue cannot form a vertical rectangle, so the upper-right optional area must be covered by the clue to its left, and because the bottom cell in the central column is covered by the clue to its upper-left, the lower-right optional area must also be covered by the clue to its left. When the upper optional area is not covered by the input wire, it must be covered by the central clue forming a vertical rectangle, so the output optional areas cannot be covered by the clues to their left.

Shift. Because variable and gate outputs are on the right and gate inputs are on the left, we do not need a turn gadget, but we do need to shift wires vertically, which is done using the shift gadget. An upward shift gadget is shown in Figure 20; the downward shift gadget is that gadget's reflection across the horizontal axis. When the input wire is true, the input wire covers the left optional area, so the left clue is covered by a single cell and the right clue covers the right optional area, propagating true on the output. When the input wire is false, the left clue covers the left optional area and forces the right clue to be a 1×2 rectangle, leaving the right optional area uncovered, propagating false on the output.

Layout. Friedman's proof in [15] omits discussion of layout, but we sketch a layout algorithm here. We start with a grid embedding of the input planar Boolean circuit. We scale the grid by at least 6 so that our wire gadget fits for unit-length wires, but possibly by a greater factor if the grid embedding has long vertical segments, because our shift gadget consumes horizontal distance to move vertically.

Filling algorithm. The filling algorithm places a clue in the center of every cell that isn't part of a gadget, forcing them to be covered by single-cell areas. Filler clues could only cover area in a gadget if two cells in the gadget area are separated by one filler clue and those cells do not themselves have clues. This is avoided in all gadgets by ensuring all gadget cells that are separated by filler are separated by two or more filler cells, so only local gadget solutions are possible.

Composition algorithm. The local gadget solutions are already consistent with each other, so to form an area assignment for the entire puzzle, the composition algorithm simply takes the local gadget solutions and assigns each filler clue to the cell containing it.

Proper and complete profile tables. The profile tables are proper because they contain only proper profiles. Because the filler clues cannot cover area in the gadgets, we can verify by case analysis that the profile tables are complete (all other profiles are locally impossible). This completes the proof.

Acknowledgments

This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.890) taught by Erik Demaine in Fall 2014. We thank the other participants of that class for related discussions and providing an inspiring atmosphere.

References

1 Zachary Abel, Jeffrey Bosboom, Erik D. Demaine, Linus Hamilton, Adam Hesterberg, Justin Kopinsky, Jayson Lynch, and Mikhail Rudoy. Who witnesses The Witness? Finding witnesses in The Witness is hard and sometimes impossible. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 3:1–3:21, La Maddalena, Italy, June 2018.

2 Aaron B. Adcock, Erik D. Demaine, Martin L. Demaine, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, and Blair D. Sullivan. Zig-zag numberlink is NP-complete. *Journal of Information Processing*, 23(3):239–245, 2015.

3 Aviv Adler, Michael Biro, Erik Demaine, Mikhail Rudoy, and Christiane Schmidt. Computational complexity of numberless Shakashaka. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015)*, Kingston, Canada, August 2015.

4 Aviv Adler, Jeffrey Bosboom, Erik D. Demaine, Martin L. Demaine, Quanquan C. Liu, and Jayson Lynch. Z3-based Tatamibari solver, and figures from Tatamibari NP-hardness paper. https://github.com/jbosboom/tatamibari-solver, 2020. Includes inputs for the gadgets in this paper.

5 Addison Allen, Daniel Packer, Sophia White, and Aaron Williams. Pencils and Sto-Stone are NP-complete [paper in review]. https://www.researchgate.net/project/Computational-Complexity-of-Video-Games-and-Puzzles/update/5aa7c402b53d2f0bba57bfb8, 13 March 2018.

6 Walker Anderson, Erik D. Demaine, and Martin L. Demaine. Spiral galaxies font. In Jennifer Beineke and Jason Rosenhouse, editors, *The Mathematics of Various Entertaining Subjects (MOVES 2017)*, volume 3, pages 24–30. Princeton University Press, 2019.

7 Daniel Andersson. HIROIMONO is NP-complete. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 30–39, 2007.

8 Daniel Andersson. Hashiwokakero is NP-complete. *Information Processing Letters*, 109(19):1145–1146, 2009.

9 Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In My T. Thai and Sartaj Sahni, editors, *Proceedings of the 16th Annual International Conference on Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 216–225, July 2010.

10 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Budapest, Hungary, 2008.

11 Erik D. Demaine and Martin L. Demaine. Fun with fonts: Algorithmic typography. *Theoretical Computer Science*, 586:111–119, June 2015.

12 Erik D. Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of Shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97-A(6):1213–1219, 2014.

13 Erich Friedman. Corral puzzles are NP-complete. http://www.stetson.edu/~efriedma/papers/corral/corral.html, August 2002.

14 Erich Friedman. Pearl puzzles are NP-complete. http://www.stetson.edu/~efriedma/papers/pearl/pearl.html, August 2002.

15 Erich Friedman. Spiral Galaxies puzzles are NP-complete. https://www2.stetson.edu/~efriedma/papers/spiral/spiral.html, March 2002.

16 Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, July 1977.

17 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation.* A K Peters, July 2009.

18 Markus Holzer, Andreas Klein, and Martin Kutrib. On the NP-completeness of the NURIKABE pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on FUN with Algorithms*, pages 77–89, Isola d'Elba, Italy, May 2004.

19 Markus Holzer and Oliver Ruepp. The troubles of interior design—a complexity analysis of the game Heyawake. In *Proceedings of the 4th International Conference on FUN with Algorithms*, volume 4475 of *Lecture Notes in Computer Science*, pages 198–212, 2007.

20 Ayaka Ishibashi, Yuichi Sato, and Shigeki Iwata. NP-completeness of two pencil puzzles: Yajilin and Country Road. *Utilitas Mathematica*, 88:237–246, 2012.

21 Chuzo Iwamoto. Yosenabe is NP-complete. *Journal of Information Processing*, 22(1):40–43, 2014.

22 Jonas Kölker. Kurodoko is NP-complete. *Journal of Information Processing*, 20(3):694–706, 2012.

23 Kouichi Kotsuma and Yasuhiko Takenaga. NP-completeness and enumeration of Number Link puzzle. *IEICE Technical Report*, 109(465):1–7, March 2010.

24 Brandon McPhail. The complexity of puzzles. Undergraduate thesis, Reed College, Portland, Oregon, 2003.

25 Brandon McPhail. Light Up is NP-complete. http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf, 2005.

26 Brandon McPhail. Metapuzzles: Reducing SAT to your favorite puzzle. CS Theory talk, December 2007.

27 Nikoli Co., Ltd. タタミバリ（仮題）(Tatamibari (temporary title)). *Puzzle Communication Nikoli*, 107, 10 June 2004. See https://www.nikoli.co.jp/ja/publication/various/nikoli/back_number/nikoli107/ for a table of contents.

28 Nikoli Co., Ltd. パズル通信ニコリ > オモパリスト (Puzzle Communication Nikoli > Omopa List). https://www.nikoli.co.jp/ja/publication/various/nikoli/omopalist/, 2020.

29 Nikoli Co., Ltd. Nikoli puzzles. http://www.nikoli.co.jp/en/puzzles/, 2020.

30 Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, May 1996.

31 Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, University of Tokyo, Tokyo, Japan, January 2003.

32 Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E86-A(5):1052–1060, 2003. Also IPSJ SIG Notes 2002-AL-87-2, 2002.