The Troubles of Interior Design–A Complexity Analysis of the Game Heyawake

Markus Holzer and Oliver Ruepp

Institut für Informatik, Technische Universität München,
Boltzmannstrasse 3, D-85748 Garching bei München, Germany
{holzer,ruepp}@in.tum.de

Abstract. HEYAWAKE is one of many recently popular Japanese pencil puzzles. We investigate the computational complexity of the problem of deciding whether a given puzzle instance has a solution or not. We show that Boolean gates can be emulated via HEYAWAKE puzzles, and that it is possible to reduce the Boolean Satisfiability problem to HEYAWAKE. It follows that the problem in question is NP-complete.

1 Introduction

HEYAWAKE is one of many pencil puzzles published by the Japanese company *Nikoli Inc.* which specializes in logic games. Pencil puzzles have gained considerable popularity during recent years. The arguably most prominent example is the game of Number Place (jap. Sudoku), which first appeared as early as 1979 in an American magazine, but did not receive much attention until *Nikoli Inc.* published their version of the puzzle on the Japanese market. Being a big hit in Japan, the puzzle later became very popular around the whole world, and now the interest in other pencil puzzles is also rising.

As most other pencil puzzles, HEYAWAKE (engl. "divided rooms") is played on a finite, two-dimensional rectangular grid. Compared to most other pencil puzzles however, HEYAWAKE seems to be substantially more complicated due to its many rules. The grid is sub-divided into smaller rectangles (which are also called rooms, hence the name), and each of these rectangles may or may not contain a number. The sub-rectangles must form a disjoint partition of the whole grid. The goal of the game is to paint the cells of the board either white or black, according to the following rules:

1. Black cells are never horizontally or vertically adjacent.
2. All white cells must be interconnected. Diagonal connections do not count.
3. If a sub-rectangle contains a number, it must contain exactly that many black fields. Otherwise, any number of black cells is allowed.
4. Any horizontal or vertical straight line of white cells must not pass through more than 2 sub-rectangles.

Figure 1 shows an example HEYAWAKE puzzle and its solution. The reader is encouraged to verify that the solution is unique. Lots of puzzles are available on

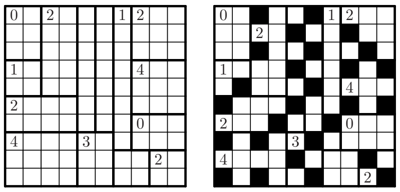


Fig. 1. Heyawake example puzzle (left) and its solution (right)

the internet, e.g., on `http://www.nikoli.com` or on `http://www.janko.at` (in German).

For a computer scientist, pencil puzzles are especially interesting from the computational complexity point of view. The probably most basic problem is finding a solution for a given puzzle, and in most cases, the corresponding decision problem ("is there a solution?") turns out to be NP-complete. Here NP denotes the class of problems solvable in polynomial time on a nondeterministic Turing machine. To our knowledge, the first result on pencil puzzles is due to Ueda and Nagao [8], who showed that Nonogram is NP-complete. Since then, a number of other pencil puzzles have been found to be NP-complete, e.g., Corral [2], Pearl [3], Spiral Galaxies [4], Nurikabe [6], Cross Sum (Jap. Kakkuro) [7], Slither Link [10], Number Place (Jap. Sudoku) [10], and Fillomino [10]. We contribute to this list by showing that Heyawake is NP-complete, too, proving the following theorem:

Theorem 1. *Solving a* Heyawake *puzzle is* NP*-complete.*

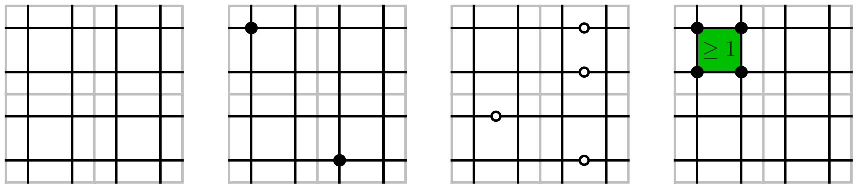
To this end, we show how to emulate Boolean circuits via Heyawake puzzles. We assume the reader to be familiar with the basics of complexity theory as contained in [5]. Hardness and completeness are always meant with respect to deterministic many-one log-space reducibilities.

2 Heyawake Is Intractable

To prove Theorem 1, we have to show that the problem in question is contained in NP, and that it is NP-hard. The containment in NP is immediate, since it is obvious that a nondeterministic Turing machine can guess a black and white pattern and check if that pattern constitutes a valid solution in polynomial time. It remains to prove the NP-hardness of the problem. We achieve this by showing how to reduce a 3SAT formula to Heyawake. We define the problem 3SAT as follows:

Instance: A finite set of Boolean variables $X = \{x_1, x_2, \dots, x_n\}$ and a finite set of clauses $C = \{c_1, c_2, \dots, c_m\}$, where each clause consists of 3 literals.

Question: If the input is interpreted in the obvious way as a 3CNF formula, is there an assignment for the variables such that the formula evaluates to true?



(a) 4×4 wire grid (b) Soldering points (c) Negation points (d) OR-gate

Fig. 2. Basic devices, simplified representation

It is well known that this problem is NP-complete [5]. In our construction, we will use a variant of it where clauses contain 4 literals instead of three. That variant is obviously NP-complete as well, and using it simplifies the construction.

Now we are ready to present our reduction. We use the following gadgets to emulate Boolean formulas:

- A two-dimensional grid of wires: Each wire in the grid will carry a Boolean value. The gadget is designed such that crossing wires will not affect each other (unless connected by a soldering point).
- Soldering points: They are used to synchronize the values between two crossing wires.
- NOT-gates: Used to invert a Boolean signal.
- OR-gates: Each of these has 4 inputs, and works in a slightly "nonstandard" way. Instead of producing another Boolean output value, it won't allow that all of the input values are "false."

Before we start to explain in detail how to emulate these gadgets through Heyawake puzzles, we want to provide the reader with a more schematic overview of our reduction. Figure 2 shows some simplified symbolic drawings for our gadgets. The basic size of our devices is chosen such that 2 horizontal and 2 vertical wires fit, device boundaries are indicated by grey bounding boxes. This means that some space is wasted, but as we will see later, it also helps to greatly simplify and clarify the actual Heyawake construction. The 4 inputs to the OR-gate are indicated by the soldering points on the corners of the gadget. Note that these soldering points are not merely decoration, but they actually function as soldering points. This means that the horizontal and vertical wires that meet at one input have their values synchronized. However, the 4 input values are of course decoupled through the OR-gate, so, e.g., the value at the lower left input may be different from the value at the lower right input and so forth.

There will be one OR-gadget for each clause, placed at a certain position in the grid, and by using the soldering point gadgets, the appropriate Boolean values can be transported to the OR-gate. Note that we don't need an AND-gate, because its functionality is already implicitly present: The only thing that the AND-operator in a 3CNF formula does is that it forces the value of **every** clause to "true," and as we explained above, our OR-gate already does that. Figure 3 shows an example construction.

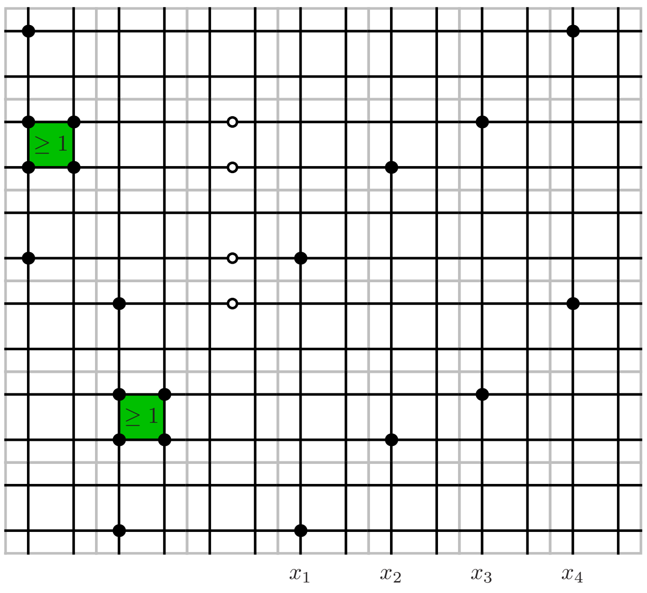


Fig. 3. Schematic construction for example formula $(x_1 \vee x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$

It is quite easy to see how this example construction can be extended to represent arbitrary formulae. For every variable that occurs in a formula, we add two vertical wires, one carrying the signal for the variable and the other is just a spacer which carries an arbitrary signal.

Another two unused vertical wires are added left of the variable wires. They are added so we can negate values on the horizontal wires that cross these vertical wires. That way, we can easily determine the original content of any clause in the input formula by just reading off the negation points in that vertical layer of the construction.

Finally, we add an OR-gadget for each clause, and we transport the Boolean variable signals to the gadget using 6 horizontal and two vertical wires. The first two gadgets are placed as shown in the example, and any additional gadget is then added two wires to the left and 6 wires above the last.

We will now present the Heyawake sub-puzzles used in our construction. Rooms that have a single unique solution have their solution pattern entered. Cells are painted light green if we know that they cannot possibly be colored black because of any of our 4 rules. Whenever we talk about coordinates of form (x, y), they have the following meaning: The upper left corner has coordinates $(1, 1)$, and the values denote first the horizontal and then the vertical position relative to that spot. Sizes of gadgets are specified as $h \times v$ with h and v denoting horizontal and vertical extent. We leave a border of undetermined cells around the sub-puzzles to indicate that they must never be regarded as stand-alone puzzles, and there will always be some kind of surrounding, like, e.g., another gadget.

The wire grid is the basis for our whole construction. An example wire grid of size 2×2 is shown in Figure 4. There are essentially three different kinds of sub-rectangles used. We explain these in detail, because most of them are also important for the other gadgets.

- 3×3 rooms with a number of 5 black fields. There is obviously only one solution for these rooms. We will refer to them as "spacer rooms," and they are basically used to decouple the solution patterns in any adjacent rooms.
- 3×4 (resp. 4×3) rooms with a number of three black fields. These are the rooms that represent our wires, thus we will also refer to them as "wire rooms." Because of the vertically (resp. horizontally) adjacent 3×3 rooms, only 3×2 (resp. 2×3) fields are left for the black cells, and thus there are exactly two possible solutions for this kind of room. Each of these solutions represents a Boolean value.
- 4×4 rooms with two black fields. These represent the crossing of wires, hence we will call them "crossing rooms." There are two possible solutions for these: Because of Rule 4, the black cells must be either in the upper left and lower right corner, or in the upper right and lower left corner. Choosing a solution for one of these rooms also determines the solution for all other rooms of this kind (because of Rule 4), they must all be filled with the same solution.

To propagate the Boolean values, we use only Rule 4. If a certain solution is chosen for a horizontal resp. vertical wire room, we have to choose the other solution for the next horizontal resp. vertical wire room. Figure 4 also shows that choosing the same solution leads to an invalid configuration, the red fields indicate a white line that spans over three rooms and thus violates Rule 4. So the solution pattern representing a certain Boolean value alternates from one wire room to the next. We still have to define which solution represents which value, but since the interpretation depends on the actual implementation of the OR-gadget, we will get to this later.

To combine several wire gadgets to form a larger grid, we can just copy the gadget such that the spacer rooms on the border coincide. If we want, e.g., to create a 4×2 wire grid from the example shown in Figure 4 (which has a size of 17×17 cells), we would need to take the 17×17 block of cells at $(3, 3)$ and copy it to position $(17, 3)$. Vertical extension works analogous.

The Heyawake puzzle corresponding to a soldering point is shown in Figure 5. The actual gadget has size 10×10 and can be found at position $(10, 10)$ in the diagram. Whenever we copy that 10×10 block over a wire crossing in a wire grid (again such that the 3×3 spacers coincide), the corresponding vertical and horizontal wires have their values synchronized. Note that we also have to modify the four adjacent crossing rooms: They are extended by one cell towards the middle of the gadget. If we leave the crossing rooms unchanged, the two black fields of the those rooms could no longer be placed arbitrarily (because of Rule 2), but a certain solution would be enforced. This solution depends on the Boolean input value of the gadget and it thus would conflict with any other soldering point gadget that has the other Boolean value as input.

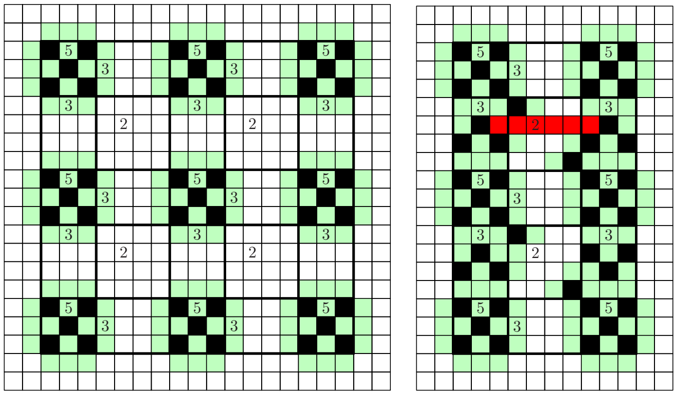


Fig. 4. Wire construction (left) and propagating values with Rule 4 (right), showing an inconsistent solution (top right) and a consistent solution (bottom right)

Note that because we need to modify the adjacent crossing rooms, combining soldering points is difficult: If we place a soldering point on a wire crossing, we cannot place another soldering point on any of the 8 neighbouring wire crossings. Also, if two soldering points are placed on the same wire with only one wire crossing in between, then the crossing room between these soldering points has to grow in both directions.

Again, Rule 4 is used to propagate information: There are four rooms of sizes 2×4 resp. 4×2 and one room of size 2×2 in the middle. Each of these rooms effectively contains a 2×2 region of cells that has to be filled with two black cells each. The information is transferred through the rooms marked with 0. All of the 2×2 regions must be filled with the same pattern, because otherwise Rule 4 is violated.

Figure 5 also contains a negation point in the right part of the diagram. The basic idea is to shrink one of the wire rooms horizontally, thus one of the adjacent crossing rooms must grow, as in the soldering point gadget. The alternating pattern of solutions in the wire rooms is interrupted by this, and the signal is inverted. The principle can also be applied on a vertical wire, but we won't make use of that. The shrinking of the right crossing room also leads to a modification in the crossing room on the left: Two black fields are enforced there due to Rule 4. This means that we need to change the number of black fields from 2 to 3 in the left crossing room. The neighboring crossing rooms on the top, left and bottom assure that the remaining black field is placed in the lower left or upper left corner of the room, which assures that the rest of the construction is not influenced in any way. Since the crossing rooms are modified, we also cannot combine this gadget arbitrarily with any other gadgets that involve modified crossing rooms. However, we can easily avoid any conflicts by simply making our construction large enough.

The final and most important gadget in our construction is the OR-gate, shown in Figure 6. Again, the gadget requires a modification to the adjacent

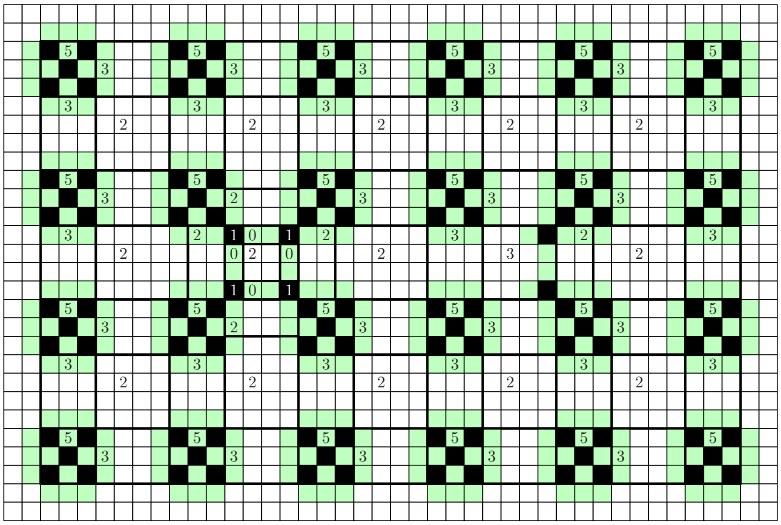


Fig. 5. Heyawake construction for soldering points (left) and negation gadget (right)

crossing rooms, and we have to be cautious about placing other such gadgets too close. The input values determine the solution of the 2×2 blocks on the corners of the gadget. A certain pattern in these rooms would lead to an invalid configuration, so the corresponding combination of input values is forbidden. Figure 7 shows that configuration: The red highlighted cells (which are actually black cells) enclose a certain area, and the white cells inside that area are disconnected from the white cells outside, which means that Rule 2 is not satisfied.

It is also easy to see that this sub-puzzle has a solution if any of the input values are changed, i.e., if not all of the corners are "closed," since the black and white pattern that has been chosen for the inner rooms remains valid independently from the input values, and all of the inner white fields get connected to the outer white fields as soon as one of the four corners "opens."

Now it also becomes clear how we have to interpret the solutions in the wire gadgets as Boolean values: The solution that leads to a "closed" corner has to be identified with "false," the other solution corresponds to "true." Looking again at Figure 3, we would like to be able to "read off" a satisfying variable assignment from a solved puzzle. It would probably be most convenient if we could just interpret the patterns that occur at the lower end of the variable wires as Boolean values. So let us examine all four input corners of the OR-gadget and find out which pattern is induced in the wire gadgets by an input value of "false." Figure 8 shows the example circuit with every variable set to "false." The signal shapes are similar to the solution shapes in the underlying Heyawake puzzle, so this diagram illustrates the following argumentation.

Since the solution pattern alternates in every wire room, the pattern that appears at the lower end of the variable wire depends on the number of wire crossings the signal passes on its way to the input corner of the OR-gate, so all

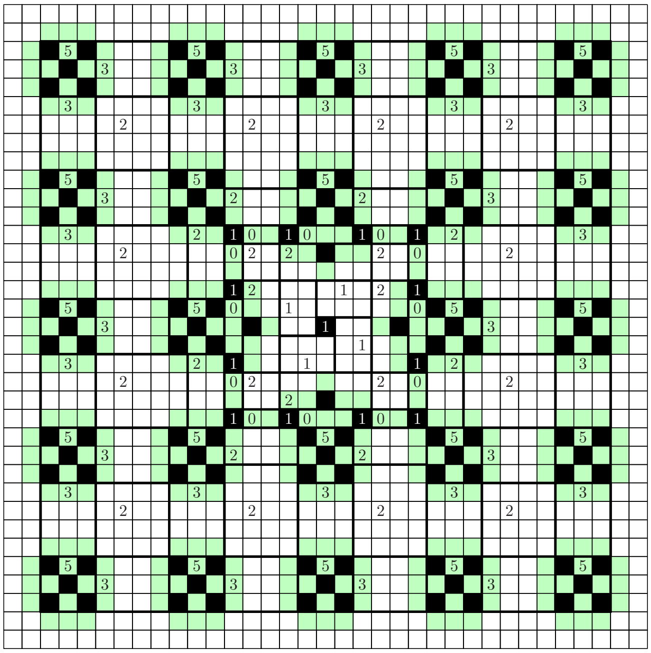


Fig. 6. The OR-gate

we need to do is count these crossings. In our example, the lower left input node of the OR-gadget is connected to the variable wire that represents x_1, and the corresponding signal passes over 6 wire crossings, the edge of the OR-gadget is not counted. From that observation, we can derive that the pattern that plays the role of "false" is the pattern that resembles an array pointing to the left. Now if we would want to connect any of the other variables to the lower left input of the OR-gate, we would have to move the corresponding soldering point from the x_1 wire horizontally to some other variable wire. Since the parity of the number of crossings is left unchanged by this process, we can conclude that the same pattern plays the role of "false" for all variables.

The next input we want to check is the lower right input. In the example, it is connected to the variable wire carrying x_2. The number of wire crossings is now odd, but also the "closing" pattern for the lower right input is inverse to that for the lower left input. This means that the same pattern as before represents the value "false" at the lower end of the variable wire. With the same argument as above, this also holds true if we connect a different variable than x_2 to the lower right input.

With analogous reasoning, we can conclude that the pattern representing "false" at the end of a variable wire is always the "left arrow" pattern, and the pattern for "true" is given by the inverse solution.

This is where the construction gets easier because of the 2×2 wire size of our gadgets. If, e.g., the variable wires were placed directly next to each other

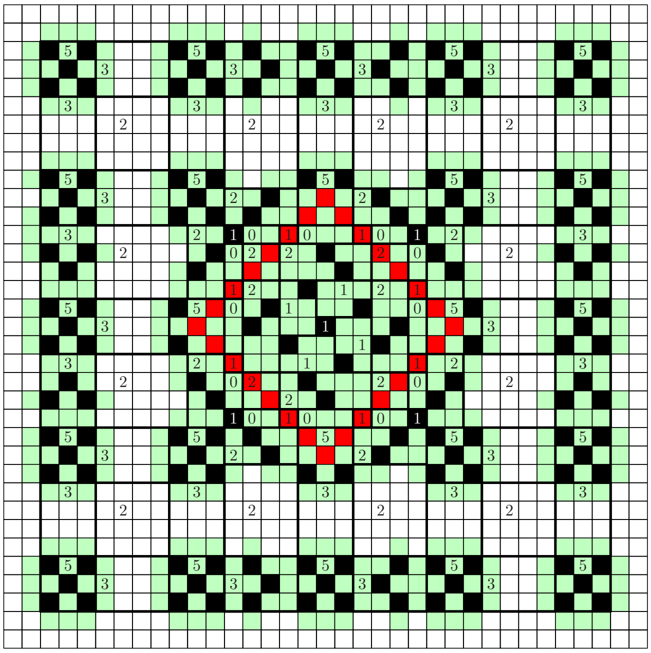


Fig. 7. The forbidden configuration

(without the unused spacing wire in between), the pattern for “false” would alternate for each variable. And if we tried to save some more space, then the interpretation of the patterns would certainly get more complicated.

The description of our construction is almost complete, but one last thing remains to be explained. We never talked about how to handle the border of the game board. If we place, e.g., a 3×3 room with five black cells directly into the upper left corner of the game grid, there is a problem: Two of the white cells (the upper and the left white cell) get isolated, Rule 2 is not satisfied. The simplest way to fix this problem is to add some border rooms to the construction. To add these rooms, we will have to increase the size of the construction by two cells on each side, so if the original construction has size $h \times v$, the size after adding the border rooms will be $(h+4) \times (v+4)$, and the original construction will be placed at $(3,3)$. The border is made up of two rooms with a size of $(h+2) \times 2$ and two rooms of size $2 \times (v+2)$. Both rooms need not contain a number. However, if we wished to abandon the feature of using blank rooms (see Rule 3), we could mark the border rooms with the number of vertical/horizontal wires used in the construction. The additional rooms are arranged such that the $2 \times (v+2)$ rooms are placed at $(1,1)$ and $(h+3,3)$, and the $(h+2) \times 2$ rooms are placed at $(3,1)$ and $(1,v+3)$. Note that they cannot possibly interfere with the rest of the construction, because all the mechanisms used in our gadgets are independent from the surroundings.

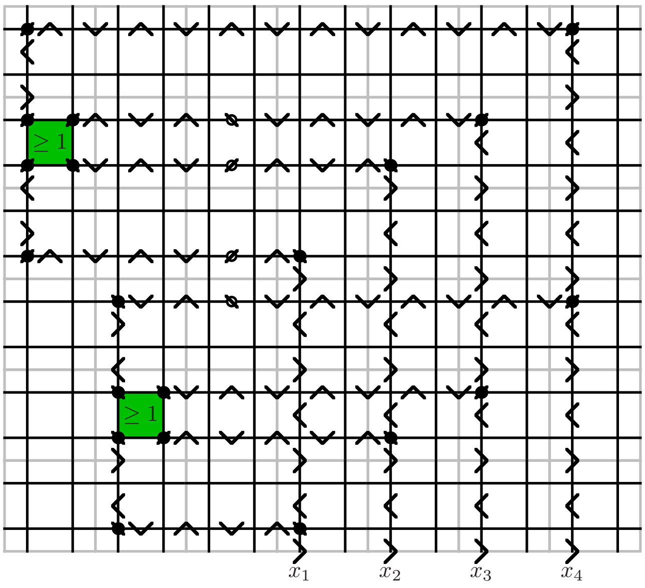


Fig. 8. Example circuit from Figure 3, with the relevant patterns of the underlying Heyawake puzzle indicated

3 A Heyawake Variant

Now that the proof of NP-completeness for the regular Heyawake ruleset is finished, it would be interesting to know if there are rules that are not essential for the NP-completeness. We will look at the variant where rule 2 is absent, which indeed turns out to be NP-complete as well:

Theorem 2. *Solving a* Heyawake *puzzle, when played with Rules 1, 3 and 4, but not with Rule 2, is* NP*-complete.*

To show NP-completeness in this new situation, we can basically reuse the construction that has been developed for the original Heyawake puzzle. All of our gadgets can easily be adapted to the new ruleset, and only slight modifications are necessary.

Figure 9 shows the new wire grid construction. It works completely analogous to the normal wire grid. The crossing rooms remain unchanged, and because of Rule 4, the two black fields will be placed in the corners of the room, just as before. Note however that choosing a solution for one crossing room no longer determines the solution for other crossing rooms, because now there is one black field between the corners. The 3×2 resp. 2×3 rooms now play the role of the wire rooms. It is obvious that these rooms have exactly two possible solution patterns, and the pattern propagates as before due to Rule 4. The new solution pattern of the wire rooms also has the convenient property that the patterns no longer alternate between wire rooms.

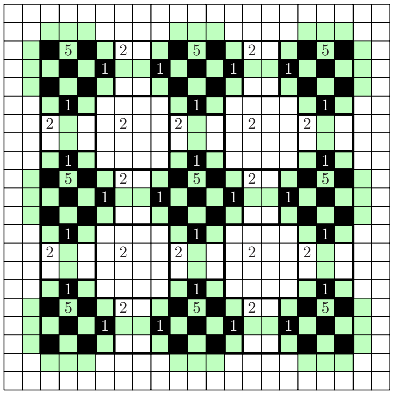


Fig. 9. The modified wire grid

The soldering point gadget is shown in Figure 10, it works exactly as before: The patterns of the two crossing wires are synchronized, because otherwise Rule 4 is not satisfied. The construction is even somewhat simpler than before: We no longer have to make modifications to the adjacent crossing rooms, because this was only necessary due to Rule 2.

The negation gadget, in contrast, is a little bit more complicated to construct under the new ruleset. Because of the modifications that have been made to the wire construction, it is not as easy as in the original construction to resize a wire room, and neither would this suffice to achieve the desired negation effect. We could use a wire room containing 3 black cells instead of 2 black cells to invert the signal, but in that case, we would need to introduce two blank rooms. Although this yields a rather simple negation gadget, we still want to get by without using the mentioned feature, so we discard the idea.

It seems substantially more complicated to emulate a negation gadget without using unspecified rooms. The best device we could find for this job is also shown in the right half of Figure 10. It is quite different from the negation gadgets discussed so far because it doesn't simply negate the signal that is carried on the affected wire. The gadget rather operates like a soldering point gadget that, instead of forcing the values on the horizontal and vertical crossing wires to be the same, forces them to be inverse to each other.

The OR-device also requires extensive modifications. In the original construction, its functionality relied entirely on Rule 2, which is not available any more. But fortunately, a simple counting mechanism can also be used to achieve the desired effect, and Figure 11 shows the resulting gadget. The idea is as follows: In the middle of the device, there is a big room that is marked with the number 9. A certain input value forces two black fields into this room, while the other value forces only one black field. The pattern that forces two fields corresponds to the value "false," and the other one corresponds to "true." If all input values are "false," we would need to put at least 10 fields into the big room, but there

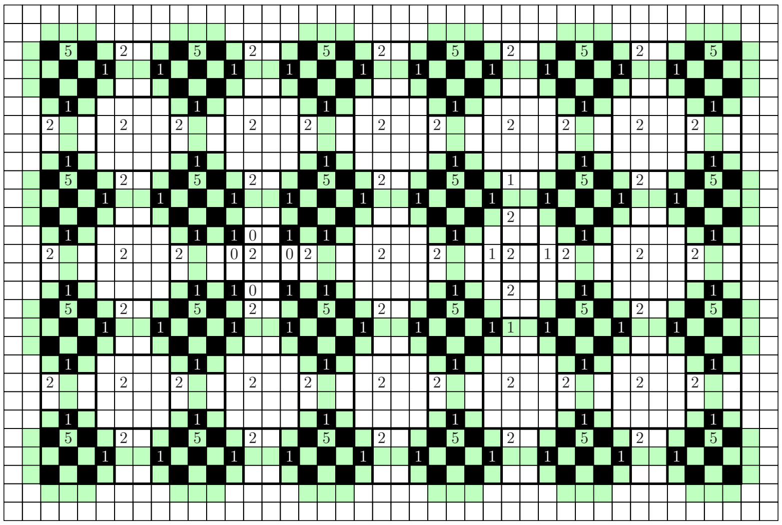


Fig. 10. Modified soldering point (left) and "negating soldering point" (right)

are only up to 9 allowed. If any of the other input values are "true" however, a solution exists. So all in all, the gadget works just like its counterpart in the original construction did.

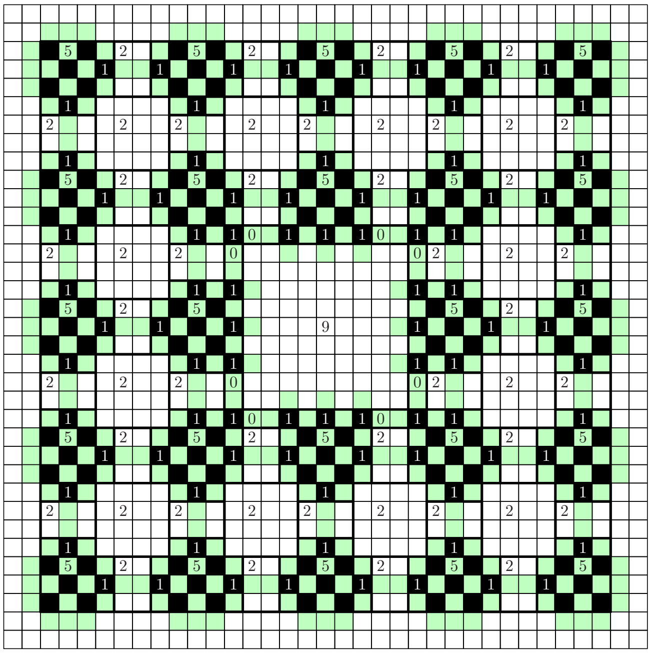


Fig. 11. OR-gadget

Now we finally have all the devices needed to carry out the overall construction. There are some minor differences, but the idea basically remains the same. Instead of dedicated negation gadgets, we now use the new "negation and soldering point" gadgets, and it is obvious that this works just as well: Every value is routed through at least one soldering point on its way to the OR-gadget, and if the value must be negated, then we simply use the negating soldering point instead of a normal soldering point.

Also, the wire room patterns have to be interpreted slightly different: Because the pattern does not alternate from room to room any more, the parity argument used in the original construction is not valid anymore. This means that it is not as easy as before to "read off" the variable values from a Heyawake solution. By looking at the input corners of all OR-gates, we can determine the pattern in the wires that induces the "false" pattern at the OR-gate. Note that a pattern that corresponds to "false" in one corner of the gadget may correspond to "true" at another corner. So we may arbitrarily choose the values that the wire room patterns represent, and then we can use negation gadgets to make sure that everything works as intended.

4 Conclusion

In this paper we have shown that the game of Heyawake is NP-complete by reducing the Boolean Satisfiability problem to the problem under consideration. We also examined a slight variation of the original puzzle and showed it to be NP-complete, too. Thus, the rule of connectivity on white cells has been shown to be artifical in the sense that it does not add to the complexity of the game.

There is another problem that has not been discussed so far, but commonly arises in puzzle making practice: Determining whether a puzzle has a unique solution or not. For a problem that can be classified as NP-complete, it is quite common that its counting variant can be classified into the class #P(see [9]). It is obvious that computing the number of solutions to a Heyawake puzzle is #P-complete: The counting variant of 3SAT is #P-complete, and we can easily determine a correspondence between the number of solutions of a 3SAT instance and the corresponding Heyawake puzzle. For each unused wire, we have to multiply the number of solutions by 2, and there are two possible solutions for the wire rooms. Looking at our example from Figure 8, there are 6 vertical and 4 horizontal unused wires, and combined with the two solutions that can be chosen for the wire rooms, we have to multiply the number of solutions of the underlying SAT instance with $2 \cdot 2^{6+4}$ to get the number of solutions that the Heyawake instance will have. It is possible to make things easier by improving our result such that there is a one-to-one correspondence between solutions.

To achieve this, we have to verify that all gadgets have the property of being completely determined by their input values, and we need some way to fix the solutions in unused wires, and in the crossing rooms. The first requirement is met all of our gadget, which is obvious in all cases except for the OR-gate. But as it turns out, even though we have not mentioned it before, the OR-gadget actually

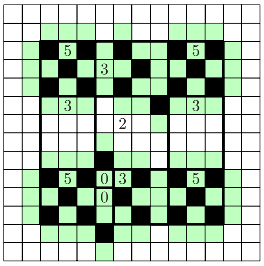


Fig. 12. Modified wire room that forces a wire value and a crossing room solution at the same time

does have that property. This can be verified by trial and error as follows: It is clear that the the inputs are completely determined, and thus the only possibly problematic part is the interior, i.e., the four rooms that are arranged around the 1 in the middle, and the four adjacent rooms that contain a 2. Each of the rooms containing a 2 has only 4 possible solutions, and if we try to solve the gadget starting with one of these solutions, a conflict will arise in all cases except for one solution.

Forcing a value in an unused wire is also quite easy: Let us look, e.g., at the lowest wire room of a vertical wire. If we simply split up that room, such that it is partitioned into one room of size 3×3 and one of size 1×3. We mark the latter room with a 0 (just because we do not want to use blank rooms), and with this we will have forced a certain pattern into the other room, due to Rule 4. The technique works completely analogous for horizontal wires.

To force a specific solution for a crossing room, we further modify the wire room at the bottom of an unused wire that has already been modified as described. We split the 1×3 room again to get a 1×1 and a 1×2 room, both marked with a 0. Because of Rule 4, this will force black fields above and below the two rooms. But this means that a black field is enforced in one corner of a crossing room, and so the whole room is determined, and in turn all other crossing rooms. The other black field will be forced in one of the border rooms, but that is no problem if we increase the number contained in that room by 1. Figure 12 shows a diagram illustrating the overall idea.

Using this modified construction, we can also derive a somewhat different, more adequate result: Checking whether a Heyawake puzzle has a unique solution is complete for the class US, which is the class of sets of type $\{ x \mid f(x) = 1 \}$, for some $f \in \#\mathsf{P}$, that has been introduced and studied in [1]. The uniqueness result is more adequate since we are not really interested in knowing the number of solutions of a Heyawake instance, as long as we know whether the solution is unique or not.

There remain some open questions: There are other rulesets for which it is unknown whether the problem remains NP-complete or not. Furthermore, it would be interesting to know if Heyawake remains NP-complete if there is a restriction on the values of the numbers in the rooms. Our original construction can be done using only numbers from 0 to 3, if the spacer rooms are divided into several single cell rooms containing the numbers 0 and 1 to imitate the spacer pattern. We don't know if this result is optimal, or if the problem remains NP-complete if the range of used numbers is reduced further.

References

1. Blass, A., Gurevich, Y.: On the unique satisfiability problem. Information and Control 55(1-3), 80–88 (1982)
2. Friedman, E.: Corral puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723 (2002)
3. Friedman, E.: Pearl puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723 (2002)
4. Friedman, E.: Spiral galaxies puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723 (2002)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co, New York, NY, USA (1990)
6. Holzer, M., Klein, A., Kutrib, M.: On the NP-completeness of the NURIKABE pencil puzzle and variants thereof. In: Ferragnia, P., Grossi, R.: (eds.). In: Proceedings of the 3rd International Conference on FUN with Algorithms, pp. 77–89, Island of Elba, Italy, Edizioni Plus, Universitá di Pisa (May 2004)
7. Seta, T.: The complexities of puzzles, cross sum and their another solution problems (asp). Senior thesis, Univ. of Tokyo, Dept. of Information Science, Faculty of Science, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan (February 2001)
8. Ueda, N., Nagao, T.: NP-completeness results for NONOGRAM via parsimonious reductions, Technical Report TR96-008, Dept. of Computer Science, Tokyo Institute of Technology (1996)
9. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)
10. Yato, T.: Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, Univ. of Tokyo, Dept. of Information Science, Faculty of Science, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan (January 2003)