Light Up is NP-complete

Brandon McPhail

February 28, 2005

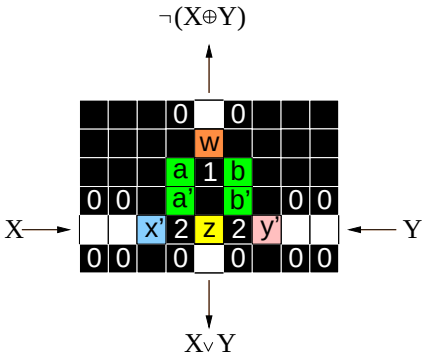

Figure 1: An OR/XNOR gate for our encoding of logic circuits as a Light Up puzzle.

Abstract

Light Up is one of many paper-and-pencil puzzles recently popular in Japan (and now elsewhere). The question, "Is this Light Up puzzle solvable?" turns out to be very hard to answer in general. We devise a polynomial-time reduction from Circuit-SAT to Light Up to prove that Light Up is NP-complete. We introduce Light Up and illustrate how Boolean circuits can be encoded as Light Up puzzles. These Light Up circuits are constructed from a collection arrangeable circuit gadgets, like the one in Fig. 1.

1 Introduction

Light Up is one of many popular *pencil-and-paper* puzzles originating in Japan. Most, if not all, paper-and-pencil puzzles consist of a grid of cells, a

list of rules, and some initially specified cells. The objective is to complete the puzzle by shading in sections of the grid in compliance with the rules. Yato[23] provides the following list of popular paper-and-pencil puzzles:

- Nurikabe[13] *
- Nonogram[19] (or Paint-by-Numbers) *
- Slither Link[22] *
- Cross Sum[17] (or *Kakkuro*) *
- Number Place[17] (or *Sudoku*) *
- Heyawake

Those marked with a * are known to be NP-complete. According to Ueda and Nagao[19], Nonogram was the first paper-and-pencil puzzle to be proved NP-complete, although various NP-completeness results have been found for paper-and-pencil puzzles since then[23]. By constructing a polynomial-time reduction from Circuit-SAT to Light Up, we present here a new NP-completeness result for paper-and-pencil puzzles.

2 How to play Light Up

Perhaps the best way to learn how to play Light Up is to visit the website of the puzzle's creator, Nikoli, at:

`http://www.nikoli.co.jp/puzzles/32/index_text-e.htm`

Many more puzzles can be printed out or played online at:

`http://www.puzzle.jp/letsplay/play_bijutsukan-e.html`

3 The decision problem

Light Up puzzles seem hard to solve. We characterize the decision problem for Light Up as, "Given a Light Up puzzle, does a solution exist?"

The complexity class P comprises of decision problems we can answer in reasonable (*polynomial*) time. Those decision problems we can *verify* in

reasonable (*polynomial*) time belong to the complexity class NP. A problem is NP-hard if *any* other problem in NP can be reduced to it. A problem is NP-complete if it is in NP and is NP-hard.

Light Up is easily seen to be in NP. Given a Light Up puzzle and a placement of lights, we can quickly determine whether each of the rules has been satisfied.

4 Theorem: Light Up is NP-hard

Any problem in NP can be reduced in time polynomial in the size of the inputs to the problem of satisfying Boolean circuits (also known as Circuit-SAT). By demonstrating a polynomial-time reduction of Circuit-SAT to Light Up, it follows that any problem in NP can be reduced to Light Up, that is, Light Up is NP-hard.

We present a proof by construction, similar to the techniques used by Kaye[11], Friedman[6], and Moore and Robson[14]. Our goal is to model the properties of a Boolean circuit using only the rules of Light Up. Given a Boolean circuit, we will construct a corresponding circuit on a Light Up board in time polynomial in the number of squares in the puzzle grid.

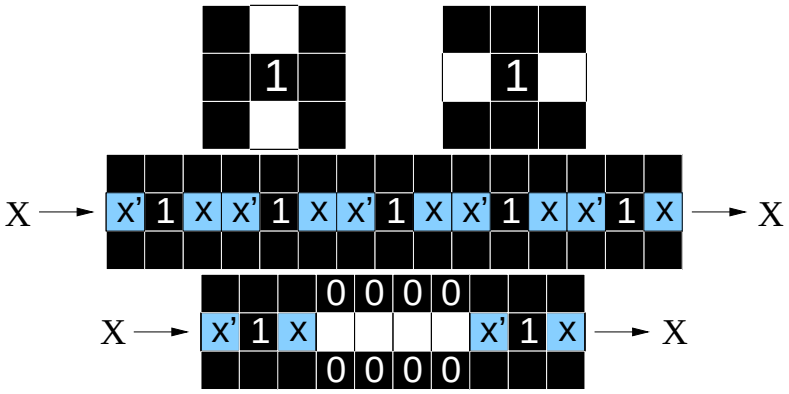

Figure 2: The wire tile (top) allows only two possible places for a light. We can string together wire tiles to propagate a Boolean signal (middle). Note that we can also "stretch" a wire tile by inserting arbitrarily long regions surrounded by zeros (bottom).

5 Proof by construction

First, we construct a grid large enough to contain our circuit. Unless otherwise specified, all cells in our grid will be black. As a Boolean circuit consists of distinct gadgets, so too will our Light Up puzzles be divided into separate parts. We refer to each part or gadget as a *tile*.

5.1 Wire construction

All of our circuits will be constructed on a simple tiling of our gadgets that is consistent with the rules of Light Up.

The **wire tile** has one of two possible states. If the x cells are assigned lights, we say the state is **true**. If instead the x' cells are assigned lights, we say the state is **false**. We can string these tiles together to propagate this *truth assignment*. Note that lining our wires up with the other gadgets in our grid-based circuit is easy, since, as demonstrated in Fig. 3, we can stretch and even bend the wires as we see fit.

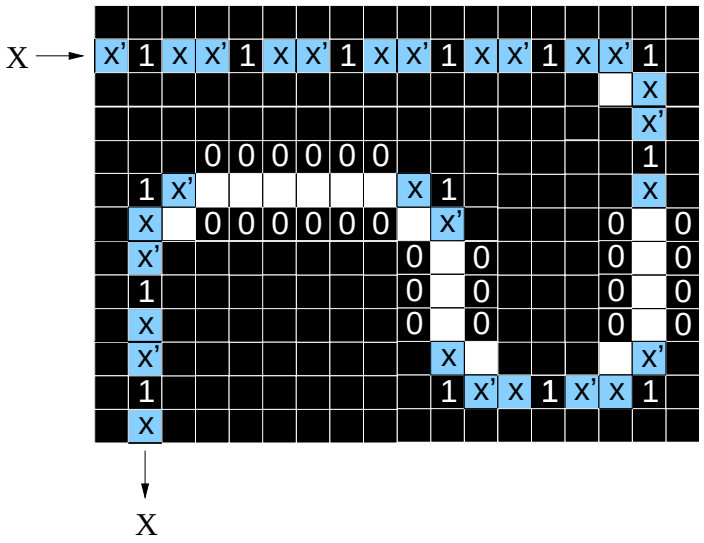

Figure 3: By introducing a set of *corner* tiles, we can bend our wires to take more interesting and useful paths.

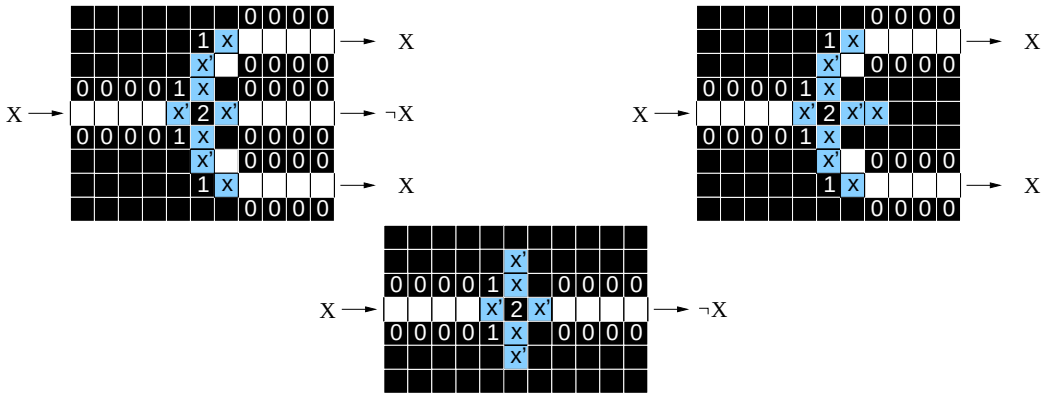

Figure 4: The branch/NOT gate (top left) allows us to split a wire into a signal and its complement. We can modify the tile in Fig. 4 to produce a NOT gate (bottom) and a branch gate (top right).

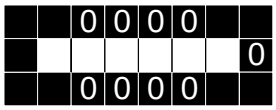

Figure 5: We can fix terminate both ends of this wire to form a complete Boolean circuit. The **0** on the right forces us to place a light bulb at the beginning of the wire on the left; we have found a satisfying truth assignment for this circuit.

5.2 Assigning values

A Boolean circuit is satisfiable if and only if our Light Up puzzle has a solution. To ensure this bijection, we fix the final output wire to **true**. The input wires may take on different values, and if we find any placement of lights for them that results in a solution, this will correspond to a satisfying truth assignment for our circuit.

Fig. 5 shows a complete Boolean circuit corresponding the Boolean expression x consisting only of a single literal. By capping the right end of the wire with a **0**, we've actually forced the final output to be true.[1] The only satisfying assignment, of course, it to set $x :=$ true, which in our Light Up puzzle means placing a light bulb in the left end of the wire.

[1]If we had capped the wire instead with a **1**, we would have forced the final output of the wire to be false.

5.3 Branch and NOT gates

We'd like to split our wires to allow the output of one gadget to form the input for multiple other gadgets. The gadget in Fig. 4 splits our wire into three wires, but the signal is flipped in the middle outgoing wire. We'll call this our **branch/NOT** gate. If we cap the middle outgoing wire of the branch/NOT gate, we get a **branch gate**. If we instead cap the top and bottom outgoing wires of the **branch/NOT** gate, we are left with just a **NOT gate**.

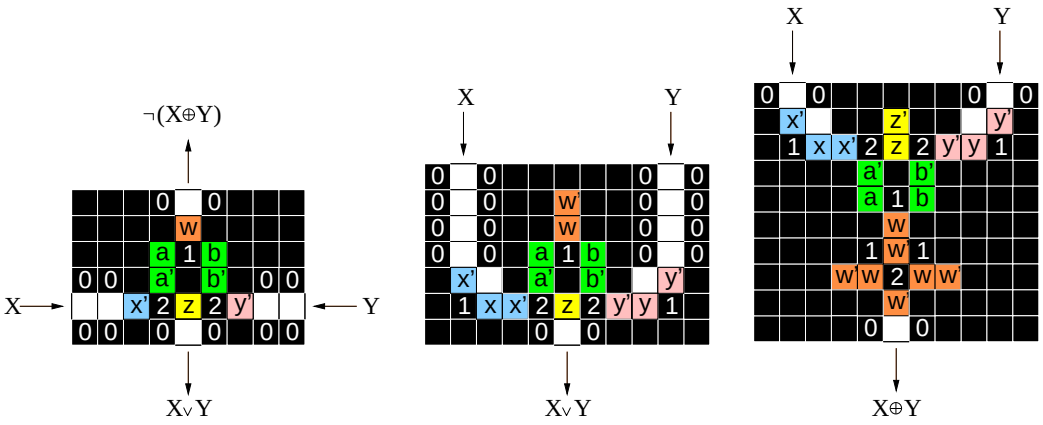

Figure 6: Given inputs x and y, the OR/XNOR gate (left) outputs both the exlusive NOR and inclusive OR of the inputs. Capping one of these outputs produces an OR gate (middle) or an XOR gate (right).

5.4 The OR gate

The **OR/XNOR gate**, like the branch/NOT gate, is a two-for-one logic gate. Given two input wires (we'll label them x and y), the OR/XNOR gate outputs *both* the exclusive NOR $\neg(x \oplus y) = (x \wedge y) \vee (\neg x \wedge \neg y)$ and the inclusive OR $x \vee y$. We can of course cap one of these two outgoing wires to transform the OR/XNOR gate into just an XNOR gate or just an inclusive OR gate.

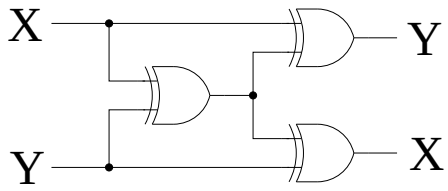

Figure 7: From XOR gates like the one in Fig. 6, we can construct a wire crossing.

5.5 Important details

Our circuit lies in a plane, so we need to either describe explicitly how wires should cross without interacting or show that wire crossing aren't necessary. As it turns out, we already have sufficient gadgetry to build a wire crossing. From an XNOR gate and a NOT gate, we can build an XOR gate. We can then use 3 branch and 3 XOR gates to allow two wires to cross.

5.6 An example construction

At this point, we have enough gadgetry to "embed" all possible Boolean circuits in Light Up puzzles. As an example, a satisfying assignment can be found for the Boolean expression

$$\neg x \vee ((x \wedge y) \vee z)$$

if and only if the Light Up puzzle in Fig. 5.6 has a solution.

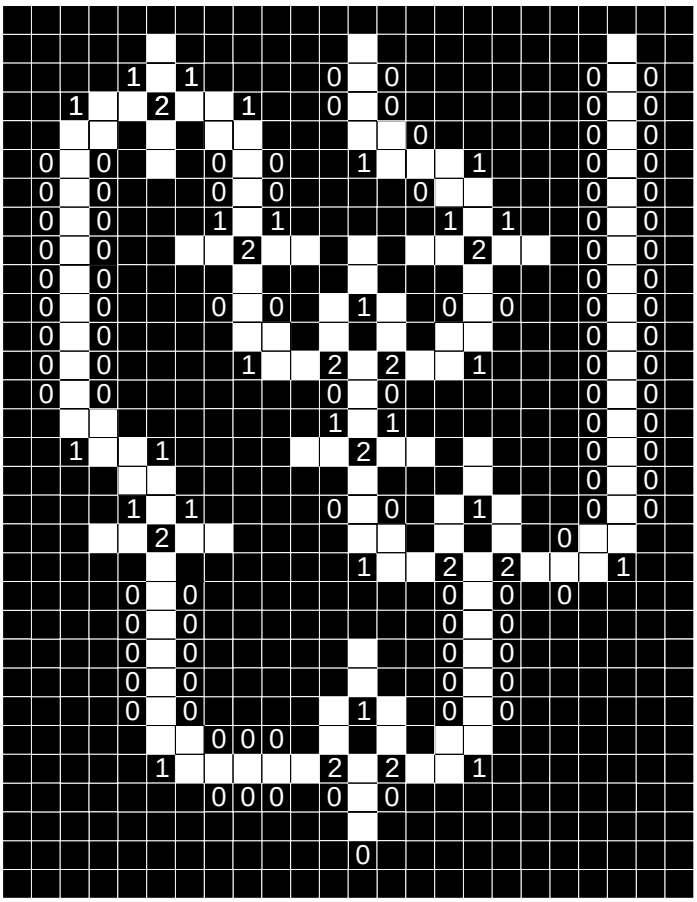

Figure 8: The Boolean expression $\neg x \vee ((x \wedge y) \vee z)$ is satisfiable if and only if this puzzle has a solution.

References

[1] L. Auslander and S. Parter. On Imbedding Graphs in the Sphere. *J. Math. Mechanics*, (10):517–523, 1961.

[2] Therese C. Biedl, Erik D. Demaine, Martin L. Demaine, Rudolf Fleischer, Lars Jacobsen, and J.Ĩan Munro. The Complexity of Clickomania, July 2000. preprint.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Comput-*

ing, pages 151–158, New York, New York, 1971. Association for Computing Machinery, ACM Press.

[4] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 1(10):41–51, 1990.

[5] Erik D. Demaine, Robert A. Hearn, and Michael Hoffman. Push-2-F is PSPACE-Complete, August 2002.

[6] Erich Friedman. Spiral Galaxies Puzzles are NP-complete. Technical report, Stetson University, 2000.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[8] John P. Hayes. *Digital System Design and Microprocessors*. McGraw-Hill, 1984.

[9] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[10] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, New York, New York, 1972. Plenum Press.

[11] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[12] Richard Kaye. Some Minesweeper Configurations. Technical report, The University of Birmingham, August 2000. http://for.mat.bham.ac.uk/R.W.Kaye.

[13] Brandon McPhail. The complexity of puzzles: NP-completeness results for Nurikabe and Minesweeper. Reed College, 2003. Undergraduate Thesis.

[14] C. Moore and J.M. Robson. Hard Tiling Problems with Simple Tiles. *Discrete & Computational Geometry*, 26(4):573–590, 2000.

[15] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[16] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[17] Seta Takahiro. The Complexities of Puzzles, Cross Sum, and their Another Solution Problems (ASP). The University of Tokyo, 2001. Undergraduate Thesis.

[18] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42(230):230–265, 1936.

[19] Nobuhisa Ueda and Tadaaki Nagao. NP-completeness Results for NONOGRAM via Parsimonious Reductions. Technical report, Tokyo Institute of Technology, 1996.

[20] L.G. Valiant and V.V. Vazirani. NP Is As Easy As Detecting Unique Solutions, 1985.

[21] Thomas Ryan Wilson. NP Completeness: Why Some Problems Are Hard. Reed College, 1995. Undergraduate Thesis.

[22] Takayuki Yato. On the NP-completeness of the Slither Link Puzzle. In *IPSJ SIGNotes ALgorithms*, pages 25–32, 2000.

[23] Takayuki Yato. Complexity and Completeness of Finding Another Solution and its Application to Puzzles. Master's thesis, The University of Tokyo, 2003.