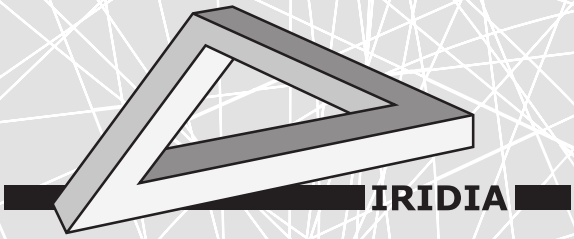

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

It is easy to Light Up in practice

N. Fagerburg, F. Mascia, and T. Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2014-004

January 2014

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2014-004

It is easy to Light Up in practice

Nils Fagerburg Franco Mascia Thomas Stützle

January 28, 2014

Abstract

Light Up is an NP-complete puzzle for which several heuristics have been proposed. In this work, we present an integer programming formulation of Light Up and show that the instances that have been tackled in the literature can be solved in fractions of a second in this formulation.

1 Introduction

Light Up is a puzzle developed by the Japanese company Nikoli in 2001 under the name *Akari* [7]. An instance of the puzzle consists of a rectangular grid containing black and white cells. The aim of the game is to place lights in some of the white cells to light up the whole board. Lights illuminate cells vertically and horizontally (not diagonally) until the beam meets a black cell or the edge of the board. Some of the black cells may contain *clues* in the form of numbers. Cells with a clue must have exactly the number of orthogonally adjacent lights as indicated by the clue. A further constraint is that a light may not illuminate another light. An example instance of the problem is given in Figure 1.

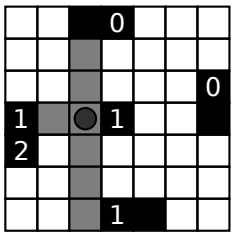

Figure 1: An example puzzle and an illustration of the cells lit by a light. The light is indicated by the circle, while the cells it illuminates are indicated by the shaded cells. (7×7 Easy instance # 689861 from [8] (redrawn)).

In 2005, McPhail proved that Light Up is NP-complete via a reduction from Circuit-SAT [4]. Several papers followed by proposing heuristic approaches [2, 6, 5] and answer set programming or constraint programming [3].

In Section 2, we formulate Light Up as an integer programming problem. In Section 3, we show that the instances that have been tackled so far in the literature can be solved in fractions of a second on a laptop computer. We draw our conclusions in Section 4.

2 Formulation

To formulate Light Up as an integer programming problem, we start by defining a decision variable $x_{i,j}$ for every white cell on the board at coordinate (i, j). Setting the value of a decision variable to 1 means that the cell is assigned a light; a value of 0 means that the cell is empty. We do not generate decision variables for black cells; alternatively, one could define a decision variable for each cell and fix the value of those associated to black cells to 0. The goal is to find an assignment to the variables that satisfies the constraints that implement the rules of the game.

We define a *group* G_k as a set containing a continuous series of white cells in a row or in a column. Therefore, each white cell belongs to exactly two groups (possibly containing only one cell): a row group and a column group. Groups are used to formulate constraints and implicitly encode if the cells are lit by a light in another cell. We then give the following integer programming formulation:

$$\text{minimize } \sum x_{i,j}$$

$$\text{subject to:}$$

$$\sum_{x_{ii,jj} \in G_k} x_{ii,jj} \geq 1 \quad \forall G_k : x_{i,j} \in G_k \;\; ; \;\; \forall x_{i,j} \tag{1}$$

$$\sum_{x_{i,j} \in G_k} x_{i,j} \leq 1 : \forall G_k \tag{2}$$

$$\sum_{\substack{k \in \{-1,0,1\} \\ l \in \{-1,0,1\} \\ k^2 \neq l^2}} x_{i+k,j+l} = n : \forall (n, i, j) \in N \tag{3}$$

$$x_{i,j} \in \{0, 1\} \;\; \forall x_{i,j}$$

In Light Up, the goal is to find a feasible solution, however, an integer programming formulation requires an objective function that has to be maximized (or minimized). In our case, we minimize the number of lights and therefore we actually solve a harder problem. Equation 1 ensures that each white cell is lit by making sure that at least one of the cells in its two groups contains a light. Equation 2 ensures that no two lights can see each other by guaranteeing that each group contains at most one light. Finally, Equation 3 ensures that each clue cell has the correct number of neighbouring lights.

3 Experimental evaluation

For the experimental evaluation, we transformed Light Up instances into their integer programming formulation, and then solved these with SCIP version 3.0.0 [1]. All experiments have been carried out on a 2.4 GHz Intel Core 2 Duo MacBook laptop with 4 GB of RAM running OS X version 10.6.8.

We use instances intended for humans from the `www.puzzle-light-up.com` website [8]. These instances were kindly shared with us by the authors of [5] and were the ones used in their paper. In the benchmark set there are ten 7×7

instances; five instances each for sizes 10×10, 14×14, 20×20, and 25×25; and six 40×30 instances.

Table 1 shows for various problem sizes the average number of variables and constraints necessary to represent the instances as integer programs. It also shows the average time in seconds for solving such instances. All instances are solved in fractions of a second on our machine during SCIP preprocessing steps. The largest instances took approximately 0.096 seconds on average including the translation time. The times obtained are orders of magnitude less than the times reported by heuristic approaches in the literature. Salcedo-Sanz et al. [6] used Light Up as a test problem in a paper about Hopfield networks. They tested their method on instances of size 14×14, which they solved within 20 seconds most of the time on an Intel Core 2, 2.13 GHz processor with 2 GB of RAM. Chiu et al. [3] used a pattern based approach inspired by the way humans play the game to attempt to solve instances. They tested their approach on instances of size 25×25 and solved them in 0.1377 seconds, on average on an AMD E8400 CPU with 8 GB of RAM. Rosberg et al. [5] used ant colony optimization after an initial preprocessing step similar to the patterns used in [3]. They report for 25×25 instances an average solving time of 8.7 seconds on a Pentium Dual core 2 GHz with 3 GB of RAM, while we solve the same size instances in 0.06 seconds on a similar speed CPU. Çelik et al. [2] compared answer set programming and constraint programming to solve Light Up and other grid puzzles. They report on the largest instances they tested (14×14) solving times around 1.3 seconds on a 2.1 GHz Intel Core2 Duo T6570 processor and 2,96 GB RAM, while we solve these instances in about 0.02 seconds on a similar speed CPU.

size	# var	# cons	total time	translation time	solving time
7x7	41.3	72.6	0.0265 (0.0081)	0.0025 (0.0002)	0.0240 (0.0080)
10x10	76.8	136.8	0.0284 (0.0051)	0.0044 (0.0002)	0.0240 (0.0049)
14x14	146.4	268.0	0.0280 (0.0089)	0.0080 (0.0006)	0.0200 (0.0089)
20x20	280.0	518.0	0.0531 (0.0138)	0.0151 (0.0007)	0.0380 (0.0133)
25x25	421.0	805.6	0.0595 (0.0147)	0.0215 (0.0016)	0.0380 (0.0160)
40x30	750.0	1468.2	0.0963 (0.0139)	0.0380 (0.0037)	0.0583 (0.0121)

Table 1: Given is for each instance size (column size) the average number of variables and constraints; the average total solving time (in parenthesis its standard deviation); as well as the breakdown of the average total time into the average time for translating the instance in the integer programming formulation, and the actual average time for solving the problem in SCIP.

4 Conclusions

In this paper, we showed that human-level instances of the Light Up puzzle can be efficiently solved by formulating Light Up as an integer program and using an efficient, non-commercial solver (in our case SCIP). This also shows that heuristic algorithms are not necessary for such instances, as the integer programs are solved in less than 0.1 seconds even for the largest instances. We conjecture that such human-level instances posses some structure that makes them appealing to humans but easy to solve automatically.

References

[1] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009. Available at `http://mpc.zib.de/index.php/MPC/article/view/4`.

[2] Mehmet Çelik, Halit Erdoğan, Firat Tahaoglu, Tansel Uras, and Esra Erdem. Comparing ASP and CP on four grid puzzles. In *Proc. of the 16th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA'09)*, 2009.

[3] Shih-Yuan Chiu, Shi-Jim Yen, Cheng-Wei Chou, and Tai-Ning Yang. A simple and rapid lights-up solver. In *Technologies and Applications of Artificial Intelligence (TAAI), 2010 International Conference on*, pages 440–443, 2010.

[4] B. McPhail. Light up is NP-complete. Available at `http://www.cs.umass.edu/~mcphailb/papers/2005lightup.pdf`, last accessed May 2013, 2005.

[5] I. Rosberg, E. Goldbarg, and M. Goldbarg. Solving the light up with ant colony optimization. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 566–573. IEEE Press, 2011.

[6] S. Salcedo-Sanz, E.G. Ortiz-García, Á.M. Pérez-Bellido, A. Portilla-Figueras, and F. López-Ferreras. On the performance of the LP-guided hopfield network-genetic algorithm. *Computers & Operations Research*, 36(7):2210–2216, 2009.

[7] Puzzles > Akari [Nikoli] `http://www.nikoli.co.jp/en/puzzles/akari.html`, last accessed May 2013.

[8] Light Up - online puzzle game, `www.puzzle-light-up.com`, last accessed May 2013.