Sudoku art

Tiffany C. Inglis and Craig S. Kaplan
David R. Cheriton School of Computer Science
University of Waterloo
piffany@gmail.com

Abstract

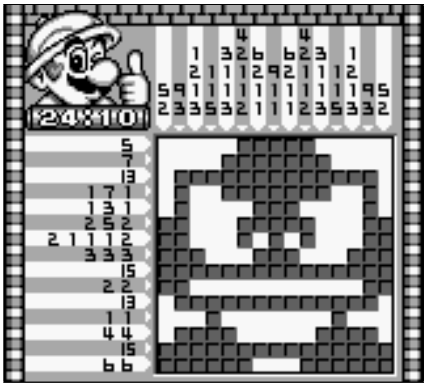
Sudoku, the popular logic puzzle, would have a greater artistic appeal if the final completed puzzle could be transformed into an image similar to nonogram outputs. To do this, we solve a mixed-integer nonlinear programming problem (MINLP) to find the Sudoku configuration that most closely represents a target image via an integer-colour mapping. This alone produces inadequate results, so we relax the problem by adding transparent regions to the target image and removing certain MINLP constraints. To produce puzzles that are feasible and interesting to players, additional components are added to avoid tedious recounting and unnecessary colouring.

1 Introduction

Sudoku, pronounced soo-DOH-koo, is a logic-based numerical puzzle which first gained popularity in 1986. The objective is simple. Given an $n \times n$ integer matrix (typically $n = 9$) with some missing entries, the player must fill in the empty cells such that each row, column and $m \times m$ ($m = \sqrt{n}$) submatrix contains each integer 1 through n exactly once. Each puzzle is created such that a unique solution exists, and can be arrived at through logical deduction. Every Sudoku solution is a Latin square, but the converse is not necessarily true since Latin squares are only subject to the row and column constraints. Figure 1a shows an example of a completed Sudoku puzzle. The black entries are given and the gray entries are filled in by the player.

4	7	2	3	1	9	8	6	5
3	6	1	2	8	5	4	7	9
8	5	9	7	6	4	3	1	2
5	9	8	1	7	6	2	4	3
1	4	6	5	3	2	9	8	7
7	2	3	9	4	8	6	5	1
2	8	7	6	5	3	1	9	4
6	3	5	4	9	1	7	2	8
9	1	4	8	2	7	5	3	6

(a) Example of a Sudoku puzzle



(b) Example of a nonogram

Figure 1: Puzzle examples

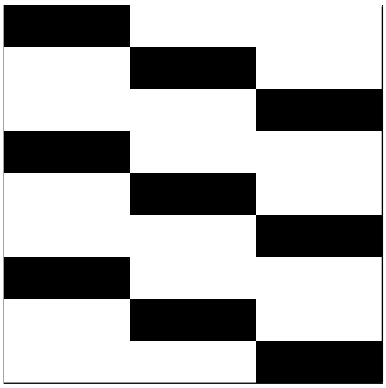
While intellectually stimulating, Sudoku puzzles do not offer much reward upon completion, other than a completed integer matrix with special properties. In contrast, a nonogram (also known as Paint by Numbers and Picross) is a logic-based puzzle that reveals a hidden image when completed. Figure 1b shows a completed nonogram taken from Mario's Picross [8]. This puzzle begins with a blank gridded canvas with strings of numbers given for each row and column indicating the ordered lengths of contiguous runs of filled cells that the solution should have. Using this information, players can deduce which cells should be filled and produce a unique solution in the form of a black-and-white pixelated image. Studies have been done to generate nonograms whose solutions depict a recognizable image [9, 2], which provides a greater incentive for finishing the puzzle. Our goal is to add a similar feature to Sudoku so that completed puzzles can be transformed into images.

Creating art subject to resource constraints has a long history. From Georges Seurat's practice of Pointillism to M. C. Escher's tesselations, artists have strived to achieve maximum aesthetics while working under certain restrictions.

In 2004, with the aid of computers, Robert Bosch described an integer programming method for constructing portraits out of dominoes [3].

We will also use an optimization approach for our problem, but before analyzing the problem in greater detail, here is a brief overview of the terminology. A *Sudoku solution*, or simply *Sudoku*, is an integer matrix satisfying the Sudoku constraints (which consist of row, column and submatrix constraints), and a *Sudoku puzzle* is a Sudoku with one or more entries removed. Each cell (i, j)—in row i and column j, counting from the top left corner—is assigned an integer (to satisfy the constraints) and a colour corresponding to that integer (to mimic the target image). An *optimal image* is a Sudoku image with minimal per-pixel difference to the target image, resulting from an *optimal Sudoku solution* and an *optimal integer-colour mapping*. **Our goal is, given a target image, find a Sudoku that most closely resembles the image when the numbers are mapped to colours.** In the next term, we will define the problem more rigourously in mathematical terms.

For example, say the 9×9 image shown in Figure 2a is the target image. Then the Sudoku in Figure 2b is an optimal Sudoku solution because by mapping the values $\{1,2,3\}$ to black and $\{4,5,6,7,8,9\}$ to white, the optimal image produced is exactly the same as the target image. In general, it is much harder to find an optimal Sudoku solution and an optimal integer-colour mapping for an arbitrary target image because the search space is extremely large.



(a) Example of a target image

1	2	3	7	8	9	4	5	6
4	5	6	1	2	3	7	8	9
7	8	9	4	5	6	1	2	3
3	1	2	9	7	8	6	4	5
6	4	5	3	1	2	9	7	8
9	7	8	6	4	5	3	1	2
2	3	1	8	9	7	5	6	4
5	6	4	2	3	1	8	9	7
8	9	7	5	6	4	2	3	1

(b) A corresponding optimal Sudoku

Figure 2: Mapping a Sudoku to an image to best represent the target image

2 Mathematical formulation

Bartlett et al. [1] describes how to write Sudoku constraints in the form of a binary integer linear program (BILP). For an $n \times n$ Sudoku, let $m = \sqrt{n}$ and define

$$v_{ijk} = \begin{cases} 1, & \text{if cell } (i,j) \text{ contains the integer } k \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Then the BILP is given by

$$\begin{aligned}
\min \quad & 0 && && \text{(objective function)} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_{ijk} = 1 && j = 1:n,\ k = 1:n && \text{(exactly one } k \text{ in each column)} \\
& \sum_{j=1}^{n} x_{ijk} = 1 && i = 1:n,\ k = 1:n && \text{(exactly one } k \text{ in each row)} \\
& \sum_{k=1}^{n} x_{ijk} = 1 && i = 1:n,\ j = 1:n && \text{(exactly one integer in each cell)} \\
& \sum_{j=m(q-1)+1}^{mq} \sum_{j=m(p-1)+1}^{mp} x_{ijk} = 1 && k = 1:n,\ p = 1:m,\ q = 1:m && \text{(exactly one } k \text{ in each submatrix).}
\end{aligned}$$

Since the objective function is identically zero, the solution set of this BILP is the set of all $n \times n$ Sudokus. To find, in this set, the Sudoku that can be mapped most closely to the target via some integer-colour mapping, we need to pose additional constraints and bias the objective function towards the target image.

Currently, an optimal Sudoku solution is represented by the binary variables x_{ijk}. To turn this into an image, let μ be the integer-colour mapping such that $\mu(k) \in [0,1]$ is the colour assigned to all cells containing integer k. Note that μ is not predetermined, but rather a result of the optimization procedure. Then it is possible to describe both the integer and the colour assigned to each cell (i,j) in terms of x_{ijk} and $\mu(k)$. Denoting the integer and the colour assigned to cell (i,j) by v_{ij} and c_{ij}, we have

$$v_{ij} = \sum_{k=1}^{n} k x_{ijk}, \tag{2}$$

$$c_{ij} = \sum_{k=1}^{n} \mu(k) x_{ijk}. \tag{3}$$

So far, v_{ij} describes the colours in the optimal image, but for this image to be optimal, it needs to be closest to the target image. The target image can be represented by an $n \times n$ matrix T such that T_{ij} is the colour of pixel (i,j). To bias the objective function σ towards the target image, define it as the Euclidean distance between the target image and the optimal image:

$$\sigma = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} (c_{ij} - T_{ij})^2} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\sum_{k=1}^{n} \mu(k) x_{ijk} - T_{ij} \right)^2} \tag{4}$$

To simplify this further, note that since each integer maps to exactly one colour, for any cell (i,j) the values $x_{ij1}, \ldots, x_{ijn}$ are all zeros except one which equals 1. Hence

$$\sum_{k=1}^{n} x_{ijk} = 1 \tag{5}$$

for all (i,j). Using Equation (5), one can reduce objective function in Equation (4) to

$$\sigma = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\sum_{k=1}^{n} x_{ijk} (\mu(k) - T_{ij}) \right)^2}. \tag{6}$$

We also know from Equation (5) that for all pairs (i,j),

$$x_{ijk_1} x_{ijk_2} = \begin{cases} 1 & \text{if } k_1 = k_2 \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

This removes all the cross terms from the squared expression in Equation (6). So we have

$$\left(\sum_{k=1}^{n} x_{ijk} (\mu(k) - T_{ij}) \right)^2 = \sum_{k=1}^{n} x_{ijk}^2 (\mu(k) - T_{ij})^2 = \sum_{k=1}^{n} x_{ijk} (\mu(k) - T_{ij})^2. \tag{8}$$

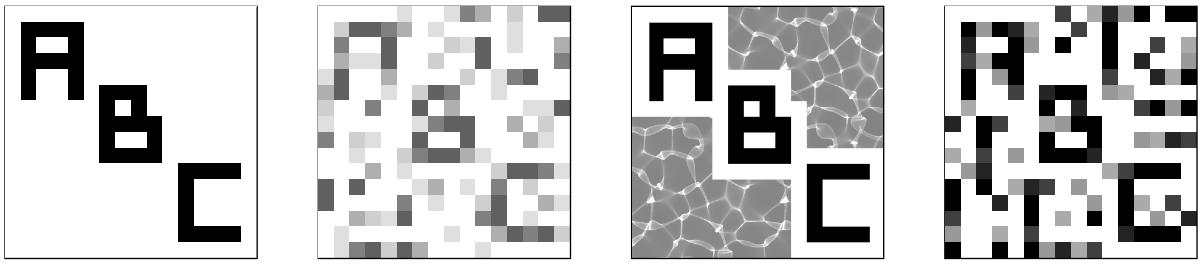
The last step is due to the fact that $x_{ijk} \in \{0,1\}$. As a result, we have

$$\sigma = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} x_{ijk} (\mu(k) - T_{ij})^2}. \tag{9}$$

In practice, σ^2 is used as the objective function since this removes the square root. Now we have a mixed-integer nonlinear program (MINLP), which can be solved using the DICOPT solver available via the NEOS Server [4, 5, 6].

3 Preliminary results

As a first attempt, a 16×16 black-and-white target image (Figure 3a) is used. Although the problem is simple, due to the rigourous Sudoku constraints, the resulting optimal image (Figure 3b) is not entirely representative of the target image. Note that it looks washed out because the target image is mostly white. To mitigate this, transparency is added to the target image. The purpose of having transparent regions is to relax the constraints, because pixels in these regions do not need count to towards the objective function. The only effect on the MINLP is that all the terms in the objective function involving the transparent pixels vanish. This way, an optimal image is allowed to contain some noise to compensate for the non-transparent pixels that must match the target image. Figure 3c shows the new target image in which the regions with caustics pattern indicate transparency. The optimal image (Figure 3d) in this case bears a much greater resemblance to the non-transparent regions of the target image.



(a) target image #1 (b) optimal image #1 (c) target image #2 (d) optimal image #2

Figure 3: Problems with Sudoku constraints, with and without transparency

These results indicate several issues. First, even though adding transparency loosens the constraints, getting a good result still requires surrounding the target image with a wide transparent border. The best we can do in terms of minimizing the amount of transparency needed is to restrict the target image to a diagonal, as done in Figure 3c. Unfortunately, this is severely limiting and the resulting optimal image is still barely recognizable. The second issue is that the optimization problem scales very poorly. Both the number of variables and the number of equations in the MINLP scale by $O\left(n^3\right)$. In practice, we cannot go beyond 16×16 in size.

4 Problem relaxation

To improve the results, it is necessary to relax the constraints of the optimization problem. By removing the submatrix constraints, we get a Latin square as the first form of a relaxed Sudoku (see Figure 4a for an example). A Latin square has the property that any permutation of its rows and columns results in another Latin square. This is not always true for a Sudoku because permuting the rows and columns may break the submatrix constraints. This property makes Latin squares more natural than Sudokus for the purpose of creating pictures because a simple translation of the target image would result in the same translation in an optimal image.

Another constraint relaxation involves reducing the number of unique integers used to fill the puzzle. Define a *p-Sudoku* as an $n \times n$ matrix containing integers from 1 to p (for some $p \leq n$) such that each row, column and $m \times m$ submatrix ($m = \sqrt{n}$) contains exactly n_i instances of the integer i ($i = 1, \ldots, p$), where $n_1, \ldots, n_p$ are integers that sum to n. For a given n, there are many possible p-Sudokus of that size. An $n \times n$ Sudoku is an example of an n-Sudoku of size $n \times n$. When constructing an optimal image, a p-Sudoku maps to only p colours. With this new construct, we can reduce the search space to the set of 2-Sudokus when dealing with 2-colour (e.g., black and white) target images, and then the resulting optimal images would automatically contain only two colours. Computationally, it is also advantageous to consider p-Sudoku solutions because the MINLP scales by $O\left(pn^2\right)$, which means much faster computations for small values of p.

1	2	3	4	5	6	7	8	9
9	1	2	3	4	5	6	7	8
8	9	1	2	3	4	5	6	7
7	8	9	1	2	3	4	5	6
6	7	8	9	1	2	3	4	5
5	6	7	8	9	1	2	3	4
4	5	6	7	8	9	1	2	3
3	4	5	6	7	8	9	1	2
2	3	4	5	6	7	8	9	1

(a) Latin square

1	2	1	2	3	2	1	2	1
1	2	1	1	2	1	2	3	2
2	3	2	1	2	1	1	2	1
1	1	2	1	2	3	2	1	2
2	1	2	1	1	2	1	2	3
1	2	3	2	1	2	1	1	2
2	1	1	2	1	2	3	2	1
3	2	1	2	1	1	2	1	2
2	1	2	3	2	1	2	1	1

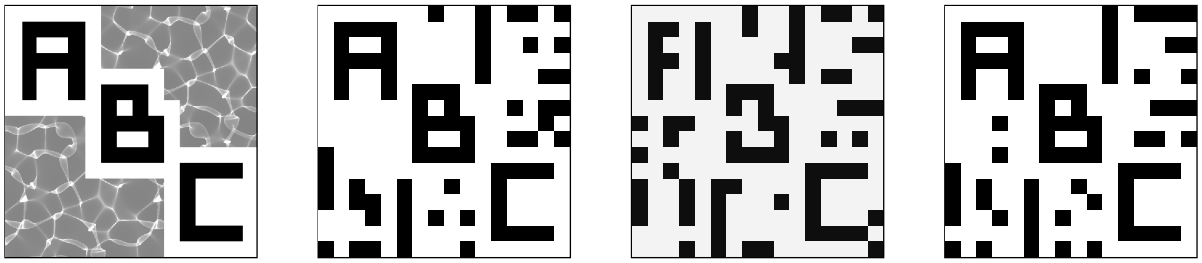
(b) 3-Sudoku

Figure 4: Examples of relaxed Sudoku solutions

We can also combine both relaxation methods to create a *p-Latin-square*, which is a *p*-Sudoku minus the submatrix constraints.

5 Relaxation results

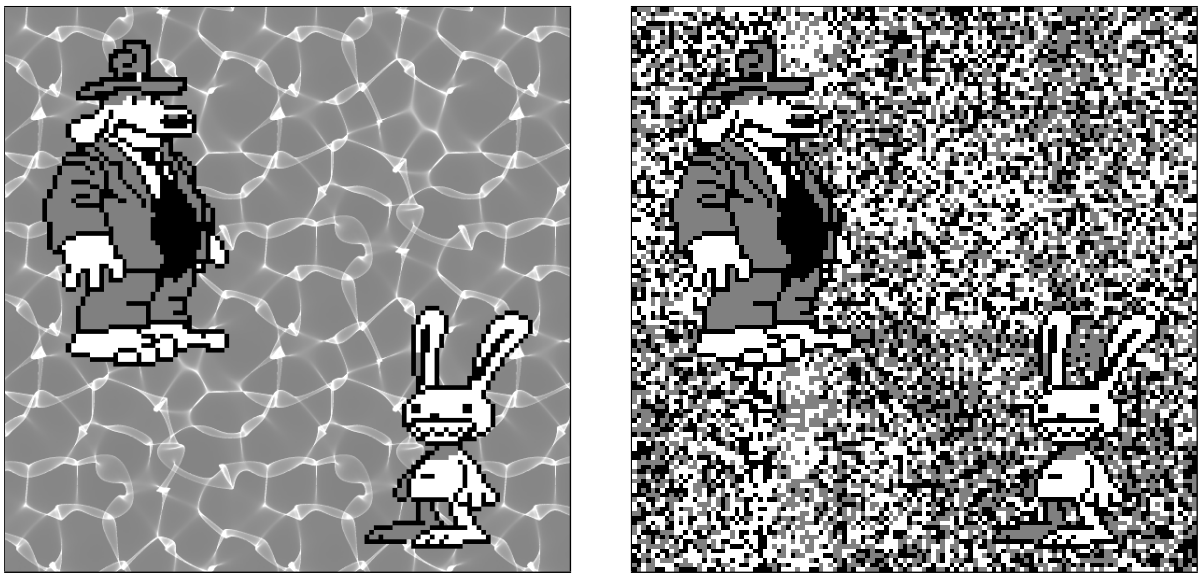
Using the target image from Figure 3c, we relax the Sudoku problem in three different ways—as a Latin square, a 2-Sudoku, and a 2-Latin-square—and find the solutions using the MINLP solver. The results are shown in Figure 5.



(a) target image (b) optimal image from Latin square (c) optimal image from 2-Sudoku (d) optimal image from 2-Latin-square

Figure 5: Problems with relaxed constraints

The optimal images from both the Latin square and the 2-Latin-square are perfect representations of the target image. That is, the objective values for both are equal to zero. Since the objective functions are Euclidean distances, this is the minimum possible value. The optimal image from the 2-Sudoku, on the other hand, has several missing black pixels. To make up for this disparity, the algorithm assigns a light gray colour to the white regions of the target image in order to minimize the objective value. Even though both the Latin square and the 2-Latin-square yield satisfactory results, the solver is much faster at finding the 2-Latin-square solution. So we test the *p*-Latin-square model again with a larger and more complex image.



(a) target image (b) optimal image from Latin square

Figure 6: Result from a 121×121 3-Latin-square

Figure 6a shows a 121×121 target image taken from a game [7]. Since it contains three colours, we set $p = 3$. The optimal image, as seen in Figure 6b, is again a perfect representation of the target image with an objective value of zero.

6 Evaluation

Using the p-Latin-square model with a small p, we are usually able to generate near-perfect optimal images for fairly complex target images in a reasonable amount of time. This is useful for creating artworks with constraints. However, if the goal is to produce a Sudoku-like puzzle, then the feasibility and gameplay logistics must be considered also.

Sudoku puzzles are typically made by first generating a complete Sudoku solution, then remove the entries one at a time such that the uniqueness of the solution is perserved after each removal. In the case of a p-Latin-square, in each step, an entry is randomly selected and checked to see if any other integer can replace it while still satisfying all the constraints. If not, that entry is removed. Otherwise, randomly select another entry and repeat the process. This continues until either enough entries have been removed or it is no longer possible to remove any more entries without creating multiple solutions.

As a feasibility test, a 36×36 2-Sudoku puzzle with 119 entries removed was given to an experienced Sudoku player. The goal was to fill in the 1's and 2's, then colour the cells containing 1's black to reveal a picture. Three problems were encountered:

1. The puzzle took more than three hours to solve, not because it was intellectually challenging but rather because it was difficult to keep track of everything in such a large puzzle.
2. The resulting image was not very distinguishable due to all the surrounding noise in the transparent regions.
3. Since fewer than 10% of the squares were removed, it was possible to reveal most of the target image without doing the puzzle.

To tackle the first problem, we may reduce the puzzle size, but as demonstrated earlier, it is necessary to allot transparent regions for noise so that the target image can be accurately portrayed in an optimal image. This means the puzzle cannot be as small as a typical 9×9 Sudoku puzzle unless the target image is even smaller than that. So instead, we will focus on how to make larger puzzles easier for the players. For instance, for each row and column in a p-Latin-square puzzle, we can provide the number of instances for each integer (similar to the nonogram setup). Players can then use this to keep track of their progress and avoid recounting at every step.

The second problem can be fixed by simply specifying which squares should be coloured after the puzzle is completed. For example, the regions containing noise can be shaded to indicate they are less important.

To fix the third problem, we choose only cells in the actual image (i.e., the non-transparent regions) to be removed. This way, players cannot reveal the target image without doing the puzzles, and they do not waste time filling in squares that constitute random noise.

Applying these changes, we arrive at a new format for our p-Latin-square puzzles as shown in Figure 7. In this example, $p = 2$ and the objective is to fill in the missing numbers so that each row and column has the same number of 1's and 2's, and then colour the cells containing 1's to reveal the target image. The gray region represents noise while the white region represents the target image. So when players complete the puzzle, they would only colour in the cells containing 1's that are in the white region to reveal the image. The two rows of numbers above the puzzle indicate how many 1's and 2's are currently filled in each column. Similarly, the two columns of numbers to the left of the puzzle correspond to the rows.

		5	5	5	1	5	5	4	4	1	5	5	5	5	5	5	5	←	1
		11	11	11	9	4	5	7	6	9	5	4	7	11	11	11	11	←	2
5	11	2	2	2	2	2	2	2	1	2	1	1	2	2	1	2	1		
5	11	2	1	1	2	2	2	2	2	2	1	2	2	2	1	1	2		
5	11	2	2	1	2	1	2	2	1	2	2	1	2	2	1	2	2		
5	8	2	2	2	2		1	1	1	1	1			2	2	2	2		
5	4	1	2	2		1						1		1	2	2	1		
3	7	2	2	1				1	2				1	2	2	2	2		
4	6	2	2	1	1			1					1	2	2	2	2		
4	5	1	1	2						2			1	1	2	2	2		
4	7	2	2	2			1			2	1		1	2	1	2	2		
2	7	2	2	2				1					1	2	2	2	2		
5	4	2	2	2		1						1		1	2	1	1		
3	8	2	2	2	2		1		1		1			2	2	2	2		
5	11	1	1	2	2	2	1	2	2	2	2	2	2	1	2	1	2		
5	11	1	2	2	2	2	1	2	2	2	2	2	2	1	1	2	1		
5	11	1	1	1	2	1	2	2	2	2	2	2	2	2	2	1	2		
5	11	2	1	2	2	1	2	2	2	2	2	1	2	2	2	1	1		
↑	↑																		
1	2																		

Figure 7: New format for a p-Latin-square puzzle

7 Future work

The changes applied to make a large puzzle more playable results in a game that is very similar to a nonogram. We would like to somehow preserve the Sudoku style of gameplay while keeping the puzzles feasible, but this may only be achievable through an interactive interface.

References

[1] Andrew C. Barlett, Timothy Chartier, Amy N. Langville, and Timothy D. Rankin. An integer programming model for the Sudoku problem. *The Journal of Online Mathematics and Its Applications*, 8, May 2006. Article ID 1798.

[2] K. Joost Batenburg, Sjoerd Henstra, Walter A. Kosters, and Willem Jan Palenstijn. Constructing simple nonograms of varying difficulty. *Pure Mathematics and Applications*, 20, 2009.

[3] Robert Bosch. Constructing domino portraits. In Barry Cipra, Erik D. Demaine, Martin L. Demaine, and Tom Rodgers, editors, *Tribute to a Mathemagician*. A K Peters, Ltd., 2005.

[4] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS Server. *IEEE Journal on Computational Science and Engineering*, 5, 1998.

[5] Elizabeth D. Dolan. The NEOS Server 4.0 administrative guide. Technical memorandum ANL/MCS-TM-250, Argonne National Laboratory, May 2001.

[6] William Gropp and Jorge J. Moré. Optimization environments and the NEOS Server. *Approximation Theory and Optimization*, 1997.

[7] LucasArts. Sam & Max Hit the Road, 1993.

[8] Nintendo. Mario's Picross, 1995.

[9] Emilio G. Ortiz-García, Sancho Salcedo-Sanz, Jose M. Leiva-Murillo, Angel M. Pérez-Bellido, and José A. Portilla-Figueras. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics*, 31(5), October 2007.