A Reasoning Framework for Solving Nonograms

K. Joost Batenburg[1] and Walter A. Kosters[2]

[1] Vision Lab, Department of Physics
University of Antwerp, Belgium `joost.batenburg@ua.ac.be`
[2] Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands `kosters@liacs.nl`

Abstract. Nonograms, also known as Japanese puzzles, are logic puzzles that are sold by many news paper vendors. The challenge is to fill a grid with black and white pixels in such a way that a given description for each row and column, indicating the lengths of consecutive segments of black pixels, is adhered to. Although the Nonograms in puzzle books can usually be solved by hand, the general problem of solving Nonograms is NP-hard. In this paper, we propose a local reasoning framework that can be used to deduce the value of certain pixels in the puzzle, given a partial filling. By iterating this procedure, starting from an empty grid, it is often possible to solve the puzzle completely. Our approach is based on ideas from dynamic programming, 2-satisfiability problems, and network flows. Our experimental results demonstrate that the approach is capable of solving a variety of Nonograms that cannot be solved by simple logic reasoning within individual rows and columns, without resorting to branching operations. Moreover, all the computations involved in the solution process can be performed in polynomial time.

1 Introduction

A *Nonogram*, also known as a *Japanese puzzle* in some countries, is a kind of logic puzzle, where the goal is to draw a rectangular image that adheres to certain row and column constraints. Usually, the image is black-and-white, although Nonograms with more than two grey values exist as well. Fig. 1 shows an example of a Nonogram. The puzzle has a rectangular shape, which is subdivided in unit cells. We will also refer to these cells as *pixels*. For each row and each column, a *description* is given. The description indicates the length of the consecutive segments of black pixels along the corresponding line. For example, the description "$1, 1$" in the first row indicates that when traversing the pixels in that row from left to right, there should first be zero or more white pixels, followed by one black pixel. Then, at least one white pixel must occur, followed by exactly one black pixel. There may be additional white pixels at the end of the line. The symbol ϵ denotes the empty description, leading to an all white line. The goal of the puzzle is to colour all pixels with either black or white, in such a way that each horizontal and vertical line is consistent with the given description.

As we shall see later, when using only information concerning single rows and columns, puzzles can often be solved partially (see the picture in the middle). For instance, one can infer that the middle pixel in the bottom row must be black. Using 2-satisfiability (2-SAT) rules we can completely solve this simple puzzle. More complicated puzzles require more sophisticated techniques, as we will also demonstrate.

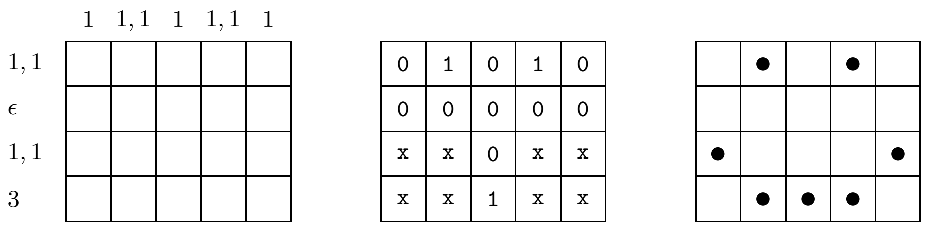


Fig. 1. A simple 4×5 Nonogram: a) original puzzle; b) partial solution (`1` = black, `0` = white, `x` = yet unknown); c) final solution (dots denote black pixels)

Nonograms can be considered as a generalization of a well-known problem in Discrete Tomography: reconstructing hv-convex sets (where the black pixels in each row and column must be consecutive). For this Discrete Tomography problem, the description for each line consists of a single number, indicating the length of the segment of black pixels along that line. The problem of reconstructing hv-convex polyominoes can be solved in polynomial time [6, 2], whereas the reconstruction problem for general hv-convex sets is NP-hard [9]. Therefore, the reconstruction problem for Nonograms is also NP-hard (and, clearly, NP-complete). In [8] this is shown through the more general concept of parsimonious reductions.

The Nonogram problem can also be related to several *job scheduling problems*, where each row corresponds to a single processor and the jobs for the processors are indicated by the row descriptions. In such scheduling problems, the type of constraints that occur in Nonograms only apply to the rows, or the columns, but not both.

There can be considerable differences in the difficulty level of Nonograms. On the one hand, the Nonograms that appear in newspapers can typically be solved by applying a series of simple logical rules, each of which considers only a single horizontal or vertical line. Later on we will refer to them as being *simple.* These puzzles will always have a unique solution. On the other hand, large random puzzles can be very difficult to solve, even using a computer, and may have many different solutions. Clearly, the fact that solving Nonograms is NP-hard indicates that not all puzzles can be solved using simple logic reasoning.

Although several implementations of Nonogram solvers can be found on the Internet (see [7] for a list of solvers), we have not found studies of this problem in the scientific literature. In [4], an evolutionary algorithm is described for solving Nonograms. Although this algorithm is quite effective at solving Nonograms, it cannot be used to find *all* solutions, if more than one solution exists.

In this paper we propose a local reasoning framework for solving Nonograms. By applying logical rules, which may involve information from several rows and columns, the value of certain unknown pixels can be deduced. By iterating this procedure, starting from an empty grid, it is often possible to either solve the puzzle completely or to determine a substantial part of the pixels. In the latter case one can distinguish between situations where there exist different solutions (that can sometimes be enumerated), and situations where one cannot infer anything anymore.

The paper is organized as follows: Section 2 introduces Nonograms in a formal, somewhat more general context; in Section 3 we show solutions to some relaxed versions (i.e., single lines, and the Discrete Tomography version); we combine these techniques into a general framework in Section 4 and Section 5, also incorporating 2-SAT rules; experiments are shown in Section 6; Section 7 concludes.

2 Notation and Concepts

We first define notation for a single line (i.e., row or column) of a Nonogram. Put $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. The symbols "0" and "1" represent the white (0) and black (1) pixels in the puzzle. In addition, we introduce a special symbol, "x", indicating that a pixel is not decided yet. Put $\Gamma = \{\mathtt{0}, \mathtt{1}, \mathtt{x}\}$. For $\ell \geq 0$, let Σ^ℓ (resp. Γ^ℓ) denote the set of all strings over Σ (resp. Γ) of length ℓ.

For describing a Nonogram, we introduce more general concepts of row and column descriptions, such that Nonograms are in fact a special case. Most of the concepts in this paper can be applied to all logic problems that follow the more general definitions.

A *description* d of length $k > 0$ is an ordered series $(d_1, d_2, \ldots, d_k)$ with $d_j = \sigma_j\{a_j, b_j\}$, where $\sigma_j \in \Sigma$ and $a_j, b_j \in \{0, 1, 2, \ldots\}$ with $a_j \leq b_j$ $(j = 1, 2, \ldots, k)$. Let D_k denote the (infinite) set of all descriptions of length k, and put $D = \cup_{k=0}^{\infty} D_k$, where D_0 consists of the empty description ϵ. A single $d_j = \sigma_j\{a_j, b_j\}$ is called a *segment description*. The perhaps somewhat confusing curly braces are used here in order to stick to the conventions from regular expressions; so, in $\sigma_j\{a_j, b_j\}$ they do not refer to a set, but to an ordered pair. We will sometimes write σ^* as a shortcut for $\sigma\{0, \infty\}$ (for $\sigma \in \Sigma$) and σ^+ as a shortcut for $\sigma\{1, \infty\}$, where ∞ is suitably large number. And finally, we put σ^a as a shortcut for $\sigma\{a, a\}$ $(a \in \{0, 1, 2, \ldots\})$, and we sometimes omit parentheses and commas; also σ^0 is omitted.

A finite string s over Σ *adheres* to a description d (as defined above) if s first has between a_1 and b_1 σ_1s (boundaries included), then between a_2 and b_2 σ_2s, …, and ends with between a_k and b_k σ_ks. Example: again take $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, and assume that the description is

$$(0\{0, \infty\}, 1\{a_1, a_1\}, 0\{1, \infty\}, 1\{a_2, a_2\}, 0\{1, \infty\}, \ldots, 1\{a_r, a_r\}, 0\{0, \infty\}).$$

This is precisely the Nonogram-type description $a_1, a_2, \ldots, a_r$ for a line (row or column). Note that it has length $2r + 1$ and can also be written as

$$(0^*, 1^{a_1}, 0^+, 1^{a_2}, 0^+, \ldots, 1^{a_r}, 0^*) = 0^* 1^{a_1} 0^+ 1^{a_2} 0^+ \ldots 1^{a_r} 0^*.$$

We denote the set of all Nonogram-type descriptions by $D_{\texttt{nonogram}} \subseteq D$. In the sequel we will concentrate on this type of description.

Suppose we have a string s over Γ. If zero or more xs are replaced with elements from Σ, the resulting string is called a *specification* of s. A specification to a string over Σ (i.e., no longer containing any "x" symbols) is called a *complete specification* or *fix*. If a string s has a fix that adheres to a given description d, s is called *fixable with respect to* d. The boolean function $\mathit{Fix}(s,d)$ is `true` if and only if s is fixable with respect to d.

A *Nonogram description* N consists of $m > 0$ row descriptions $r_1, r_2, \ldots, r_m \in D_{\texttt{nonogram}}$ and $n > 0$ column descriptions $c_1, c_2, \ldots, c_n \in D_{\texttt{nonogram}}$. A *partial filling* is a $m \times n$ matrix over Γ. The set of all these partial fillings is denoted by $\Gamma^{m \times n}$; its elements can also be considered as strings of length $m \times n$. If such a filling contains no xs, it is called a *complete filling* or *full fix*. A complete filling F *adheres* to the Nonogram description N if the ith row of F adheres to r_i (for all $i = 1, 2, \ldots, m$) and the jth column of F adheres to c_j (for all $j = 1, 2, \ldots, n$). We generalize the concepts of specification, fix and fixable in the natural way.

3 Partial Solution Methods

In this section we study two relaxations of the original problem. In Section 3.1 we confine the puzzle to a single line. In Section 3.2 we only require that the total number of black pixels in each line (i.e., row or column) adheres to its description. Clearly, any pixel that can only have a single value in all solutions of the relaxation, must also have this same value in any solution of the complete Nonogram. For both relaxations we show that such pixels can be found efficiently.

3.1 Solving a Single Line

We will now describe a recursive algorithm to decide fixability for a single line. This algorithm can be implemented by dynamic programming. First we introduce some notations. For a string $s = s_1 s_2 \ldots s_\ell$ of length ℓ over Γ we define its prefix of length i by $s^{(i)} = s_1 s_2 \ldots s_i$ ($1 \leq i \leq \ell$), so $s = s^{(\ell)}$; $s^{(0)}$ is the empty string. Similarly, for a description $d = (d_1, d_2, \ldots, d_k)$, we put $d^{(j)} = (d_1, d_2, \ldots, d_j)$ for $1 \leq j \leq k$, so $d = d^{(k)}$; $d^{(0)} = \epsilon$ is the empty description. Furthermore, let $A_j = \sum_{p=1}^{j} a_p$ and $B_j = \sum_{p=1}^{j} b_p$; put $A_0 = B_0 = 0$. We note that a string of length $\ell < A_k$ is certainly not fixable with respect to d, simply because it has too few elements; similarly, a string of length $\ell > B_k$ is not fixable with respect to d. Finally, for a given string s of length ℓ, let $L_i^\sigma(s)$ denote the largest index $h \leq i$ such that $s_h \neq \sigma$ and $s_h \neq \texttt{x}$, if such an index exists, and 0 otherwise ($\sigma \in \Sigma$, $1 \leq i \leq \ell$). We will put $\mathit{Fix}(i,j) = \mathit{Fix}(s^{(i)}, d^{(j)})$, and are interested in $\mathit{Fix}(\ell, k)$. As boundary values we note that $\mathit{Fix}(0,j) = \texttt{true}$ if and only if $A_j = 0$ ($j = 0, 1, 2, \ldots, k$); and $\mathit{Fix}(i,0) = \texttt{false}$ for $i = 1, 2, \ldots, \ell$ (by the way, these last values are never used). We clearly have $\mathit{Fix}(i,j) = \texttt{false}$ if $i < A_j$ or $i > B_j$ ($0 \leq i \leq \ell$, $0 \leq j \leq k$), as indicated above.

Our main recursion is:

$$Fix(i,j) = \bigvee_{p=\max(i-b_j,A_{j-1},L_i^{\sigma_j}(s))}^{\min(i-a_j,B_{j-1})} Fix(p,j-1) \qquad (1)$$

This holds for i and j with $1 \le i \le \ell$, $1 \le j \le k$ and $A_j \le i \le B_j$. Note that an empty disjunction is `false`; this happens for example if $L_i^{\sigma_j}(s) \ge i - a_j + 1$. For $j = 1$ we have $Fix(i,1) =$ `true` if and only if $L_i^{\sigma_1}(s) = 0$.

The validity of the recursion can be shown as follows. The last part of $s^{(i)}$ must consist of between a_j and b_j σ_js, say we want σ_js at positions $p + 1, p + 2, \ldots, i$. We then must have $a_j \le i - p \le b_j$. Also note that all elements $s_{p+1}, s_{p+2}, \ldots, s_i$ must be either `x` or σ_j; this holds exactly if $L_i^{\sigma_j}(s) \le p$. Finally, the first part of $s^{(i)}$, i.e., $s^{(p)}$, must adhere to $d^{(j-1)}$. Clearly, p must be between A_{j-1} and B_{j-1}, otherwise this would not be possible. Note that the A_j and B_j represent general tomographic restrictions, in some sense.

It is natural to implement this recursive formula by means of dynamic programming, using lazy evaluation: once a `true` $Fix(p, j-1)$ is found, the others need not be computed.

Now given a string s over Γ that is fixable with respect to a description d, it is easy to find those string elements `x` that have the same value from Σ in every fix: these elements are then set at that value. Indeed, during the computation of $Fix(s,d)$ (which of course yields `true`), one can keep track of all possible specifications that lead to a fix. In Equation (1) those $Fix(p, j-1)$ that are `true` correspond with a fix, where the string elements $s_{p+1}, s_{p+2}, \ldots, s_i$ are all equal to σ_j. Now one only has to verify, for each string element of s, whether precisely one element from Σ is allowed. In practice this can be realized by using a separate string, whose elements are filled when specifying s, and where those elements that are filled only once are tagged. Note that for this purpose lazy evaluation is not an option, since we need to examine all fixes. As an example, if the description for a five character string $s = s_1s_2s_3s_4s_5$ over $\{\mathtt{0}, \mathtt{1}, \mathtt{x}\}$ is $\mathtt{0}^*\mathtt{1}^3\mathtt{0}^*$ (cf. the bottom row from the example in Section 1), one can derive that s_3 must be equal to `1`. The algorithm that performs this operation is called *Settle*, and the resulting string s' is denoted by $s' = Settle(s,d)$.

The complexity of the computation of $Fix(\ell, k)$ is bounded by $k \cdot \ell^2$, and is in practice, especially when using lazy evaluation, much lower.

3.2 Discrete Tomography Problem

The Nonogram problem can be considered as a special case of a well-known problem from Discrete Tomography (DT), which deals with the reconstruction of a binary image from its horizontal and vertical linesums. These horizontal and vertical linesums can be easily computed from the Nonogram descriptions, by adding the segment descriptions for each line. (In the more general setting from Section 2 we get lower and upper bounds for the linesums.) Suppose that we have a partially filled Nonogram $X \in \Gamma^{m\times n}$, which we would like to extend further. Clearly, any solution of the Nonogram must also be a solution of the corresponding DT problem. The DT problem can be solved in polynomial time, even if an arbitrary subset of the image is kept fixed. It is also possible to compute the set of all pixels that must have the same value in all solutions of the DT problem in polynomial time. These pixels can be fixed immediately in the partial Nonogram solution. The paper [1] gives a constructive procedure for finding all such pixels.

Extendibility to a solution of the DT problem can easily be checked using network flow methods. We refer to [3] for the details of this model. Fig. 2a shows a simple 3×3 DT problem. We put linesums to the right of the rows and below the columns, to distinguish them from our earlier descriptions. This problem can be modelled as the transportation problem in Fig. 2b, which can be solved efficiently by network flow methods; thick arcs denote the solution. If none of the pixels are fixed, each pixel arc has a capacity of one. To fix a pixel at value $v \in \{0, 1\}$, we simply set the capacity of the corresponding pixel arc to 0 and subtract/add v to the surplus/demand at the corresponding column and row nodes. The resulting transportation problem has a solution if and only if the partial filling can be extended to a complete filling satisfying the DT constraints.

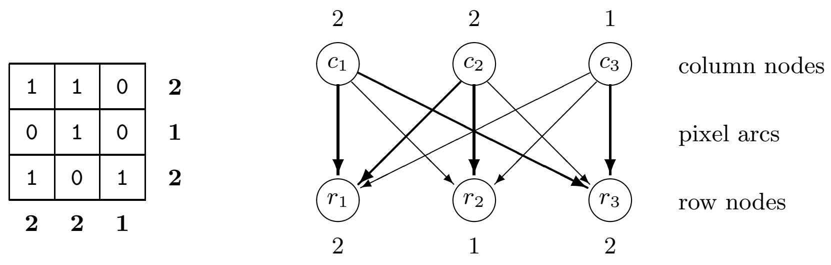


Fig. 2. a) DT problem and one of its solutions, where bold figures denote the linesums; b) associated network of the DT problem

4 Combining the Partial Methods Using 2-SAT

The method from Section 3.1 can only take into account the description of a single line. On the other hand, the discrete tomography approach from Section 3.2 can deal with all lines simultaneously, but only incorporates partial knowledge from the descriptions. We will now describe how the information from different lines, and from different relaxations of the Nonogram problem, can be combined.

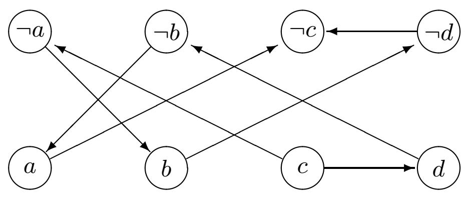
Consider the example in Fig. 3a (which is the same as that from Fig. 1). Using only the information from single lines, or from the discrete tomography problem, the values of the remaining undecided pixels cannot be derived. Four of the undecided pixels are denoted by the variables a, b, c and d respectively, which can take the values 0 (`false`) or 1 (`true`).

Using the partial solution methods, dependencies can be derived between pairs of undecided pixels. For example, on the bottom row, the description dictates that $c \Rightarrow d$ (or, equivalently, $\neg c \vee d$). Similarly, one can deduce that $c \Rightarrow \neg a$

(first column), $\neg a \Rightarrow b$ (third row) and $b \Rightarrow \neg d$. This provides us with both implications $c \Rightarrow d$ and $c \Rightarrow \neg d$, resulting in the conclusion that c must be `0`.

Note that any such implication relation between two variables can be written in one of the forms $x \vee y$, $x \vee \neg y$, $\neg x \vee y$ or $\neg x \vee \neg y$. This is the standard form of a *2-SAT clause*, see [5]. The 2-SAT problem is to decide whether or not there exists an assignment of truth values to all the variables, such that a given conjunction of such clauses is simultaneously satisfied. It can be solved in polynomial time, using the concept of a *dependency graph*, as shown in Fig. 3 for our simple example.

	1	1, 1	1	1, 1	1
1, 1	`0`	`1`	`0`	`1`	`0`
ϵ	`0`	`0`	`0`	`0`	`0`
1, 1	a	b	`0`	`x`	`x`
3	c	d	`1`	`x`	`x`



$$\begin{tikzcd}
\neg a \arrow[dr] & \neg b \arrow[dl] & \neg c & \neg d \arrow[l] \\
a \arrow[urr] & b \arrow[urr] & c \arrow[ull] \arrow[r] & d \arrow[ull]
\end{tikzcd}$$

Fig. 3. a) Partially solved Nonogram; b) (part of) its corresponding dependency graph

However, when solving a Nonogram, the goal is not to find an assignment of all variables that satisfies the 2-SAT constraints. Rather, we search for variables that must have the same truth value in *all* satisfying assignments. Assume that at least one such assignment exists. Then a variable x is `false` in all satisfying assignments if and only if there is a path from x to $\neg x$ in the dependency graph. Alternatively, x must be `true` in all satisfying assignments if and only if there is such a path from $\neg x$ to x.

This provides a polynomial-time algorithm for finding all variables that must have the same value in all satisfying assignments of the 2-SAT problem. In the example from Fig. 3a, many more 2-SAT clauses can be found from the single rows and columns, or from the discrete tomography problem.

Our procedure for combining the information from the subproblems (one for each line, and a complete DT problem) is as follows: for each pair of undecided pixels (x, y) involved in the subproblem, all four assignments are tested. For each assignment, a fixability test is performed. Each such test that returns `false` provides an additional 2-SAT clause (e.g., $x \vee \neg y$). The resulting dependency graph captures information from all subproblems simultaneously. If one considers this process as "guessing", it can also be performed in a way similar to the *Settle* operation. Indeed, when computing the *Fix* value for a line, one can keep track of all pairs of pixels, and determine those values of pairs that cannot occur.

Although the 2-SAT approach is a powerful way to combine the knowledge from different partial problems, it generally does not capture all information that is present. For example, the three character string s over $\{\texttt{0}, \texttt{1}, \texttt{x}\}$ with description $d = 0^*1^10^*$ yields rules that do not forbid the fix `000`, which is not a good fix. If one introduces clauses that can involve three variables, this leads to a clause $s_1 \neq \mathtt{0} \vee s_2 \neq \mathtt{0} \vee s_3 \neq \mathtt{0}$, which is in 3-SAT format. The general 3-SAT problem is NP-hard. Therefore, we chose the 2-SAT model, for which polynomial-time algorithms can be used.

5 Iterative Solving of Nonograms

Each relaxation of the Nonogram problem, such as the single line and discrete tomography relaxations from Section 3, can be used to deduce the value of certain pixels. By using such methods iteratively, filling in the new known pixels in each iteration, it is often possible to deduce even more pixel values. For clearness' sake, we now focus on the iterative application of the *Settle* operation from Section 3.1. It can be combined with alternative relaxations to form a more complete iterative algorithm. The *Settle* operation produces, given a string s over Γ and a description d, the string where all string elements that have the same value in every fix are set: $s \leftarrow \mathit{Settle}(s,d)$. Given $X \in \Gamma^{m\times n}$ and a Nonogram description N, we can repeat the *Settle* operation for all rows and columns of X (using the appropriate descriptions from N) until no new, previously unknown pixels are set. Note that we can use several heuristics to determine the order in which lines are examined. This operation is called *FullSettle*: $X \leftarrow \mathit{FullSettle}(X,N) \in \Gamma^{m\times n}$. If X now happens to be in $\Sigma^{m\times n}$, the puzzle is solved. Such a puzzle is called *simple*.

Note that the *Settle* operation, or rather the induced *Fix* operations, can also be used to detect certain contradictions, i.e., unsolvable puzzles. Indeed, if some line s of a proposed solution satisfies $\mathit{Fix}(s,d) = \mathtt{false}$, this solution cannot be completed.

Now, given $X \in \Gamma^{m\times n}$ and a Nonogram description N such that $X = \mathit{FullSettle}(X,N)$, we can *harvest* 2-SAT expressions, leading to a 2-SAT problem Π, and set all elements from X that have the same value in all solutions to Π as in Section 4. This operation is called *2SATSolve*: $X \leftarrow \mathit{2SATSolve}(X,N)$. The operations *FullSettle* and *2SATSolve* can be intertwined, until no further progress is made; this combination is called *Solver0*: $X \leftarrow \mathit{Solver0}(X,N)$. Again, if the resulting X happens to be in $\Sigma^{m\times n}$, the puzzle is solved. Such a puzzle is called 0–*Solvable*. Note that this whole process takes polynomial time (expressed in height and width of the puzzle).

Now suppose that $X = \mathit{Solver0}(X,N)$, but the puzzle is not solved yet. We now consider one unknown element X_{ij} from X. In a copy Y of X we *try* both $Y_{ij} = \mathtt{0}$ and $Y_{ij} = \mathtt{1}$. If for one of these $\mathit{Solver0}(Y,N)$ gives a contradiction, we know that X_{ij} must have the other value. We can, again in some heuristic order, examine all unknown pixels. Note that only those pixels that occur in a 2-SAT clause need to be examined. This procedure can be repeated, until no further progress is made, again intertwined with the use of *Solver0*. This procedure is called *Solver1*. If a Nonogram can be solved in this way, it is called 1–*Solvable*.

This process can be repeated with respect to the depth of the "tries", and it can then solve any Nonogram. In this way, we could define k–*Solvable*. Indeed, we then basically implement full backtracking. However, for our current purposes we only allow for (in depth!) one try, thereby inferring polynomial time complexity. So several tries are possible, but at each moment at most one pixel is currently "tried".

To summarize these efforts, Nonograms can be simple (if *FullSettle* solves them), 0–*Solvable* (if *Solver0* solves them), 1–*Solvable* (if *Solver1* solves them), or more complicated. Many puzzles from newspapers are simple; the example from Fig. 1 is 0–*Solvable*. Also note that *FullSettle*, *Solver0* and *Solver1* are capable of providing partial solutions. It is possible to define the difficulty level of a Nonogram as the minimum number of tries necessary to solve it. The puzzle from Fig. 1 has level 1.

There are relatively small Nonograms for which *Solver1* cannot make any progress, even though it is still possible to infer the value of certain pixels. In Fig. 4 we show an example of such a Nonogram, where one can prove that the rightmost pixel in the third row must be white, yet *Solver1* fails to infer the value of any more pixels.

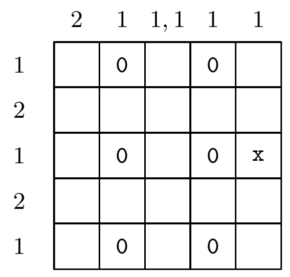


Fig. 4. Partially solved 5×5 Nonogram, where the fact that pixel `x` must be white (`0`) is hard to infer

6 Experimental Results

We will describe several experiments with the techniques from the previous sections. All considered puzzles will have at least one solution: the image that was used to construct the puzzle. We mention the observation that in our experience most puzzles from newspapers are of simple type. Although one could attribute this to the relatively small size of these puzzles, this is contradicted by the example from Fig. 1, which shows that small puzzles do not have to be of simple type. Nevertheless, the larger the puzzle, the more complicated it can be. All puzzles of simple type can be solved very fast using our proposed framework, as the operation described in Section 3.1 effectively captures all information contained in the description of each single horizontal or vertical line.

As an illustrative example for a more difficult puzzle, we mention the 30×30 Nonogram from Fig. 5. It was randomly generated with 50 % black pixels. Using only *FullSettle* just 11 pixels are found. Using *Solver0* (which only takes approximately one second on a modern PC), the puzzle is solved but for 15 pixels. One can verify that there are 6 different solutions, where it turns out that for all 15 unknown pixels both black and white can occur. This Nonogram was also included in [4], where an evolutionary algorithm was used to find one of the six solutions. A clear advantage of our reasoning framework over the algorithm presented in the former paper is that our approach finds the set of all solutions, along with a proof that there are no others. In addition, our method is much faster: seconds versus hours. On the other hand, both approaches can be considered as complementary, as the evolutionary algorithm can sometimes find a solution that cannot be deduced using our reasoning approach.

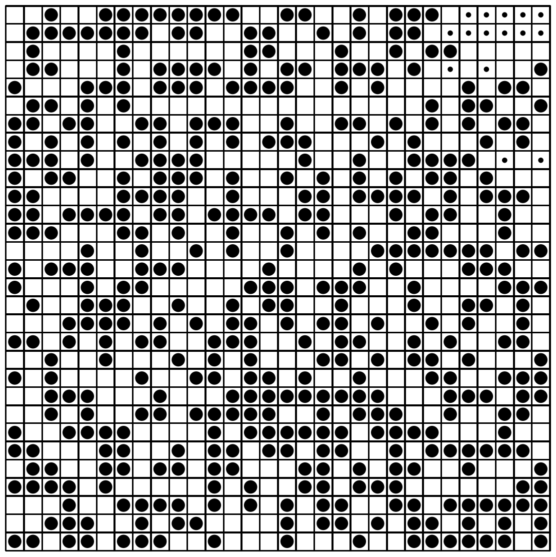


Fig. 5. Randomly generated partially solved 30×30 Nonogram, with 50 % black pixels; the 15 small dots denote the unknown pixels; 6 solutions

Most descriptions can be deduced from the figure. The description of row 1 is: $1,8,2,1,3,2$, of row 2: $7,2,2,1,1,2,2$, of row 4: $2,1,4,1,2,3,1,1,1$ and of row 9: $3,1,4,1,1,4,1$; the descriptions of the last 6 columns are: $2,4,1,1,3,1,1,1$; $1,3,1,2,2,1,1,2,3$; $1,1,1,2,2,1,1,1,1,1$; $1,1,1,3,2,1,1,3,3$; $1,1,2,1,1,4,3,1,2$; and $1,1,1,1,1,3,5$, respectively.

In Fig. 6 we see a randomly generated 40×40 Nonogram, with 881, i.e., 55 %, black pixels. In this case the puzzle has a nearly unique solution: there is only one pure 2×2 *switching component* (cf. Fig. 7), so there are 2 different solutions. The descriptions can be deduced from the figure. Again, *Solver0* is necessary: *FullSettle* finds 101 pixels.

In Fig. 8a we see the results of 7,000 runs. For each p in $\{1,2,3,\ldots,70\}$ the algorithm has been run 100 times on a randomly generated 30×30 puzzle, with p % black pixels. The piecewise linear curve connects the averages of the number of unsolved pixels (at most 900); also plotted is the standard deviation per percentage, truncated at 0 and 900. For small and large percentages the puzzles are solvable, in some cases leaving small switching components (cf. Fig. 5). Finally in Fig. 8b we plot, for each size s in $\{1, 2, 3, \ldots, 50\}$ these same quantities for randomly generated square $s \times s$ puzzles, all with 50 % black pixels. The smooth curve depicts the total number of pixels, i.e., s^2.

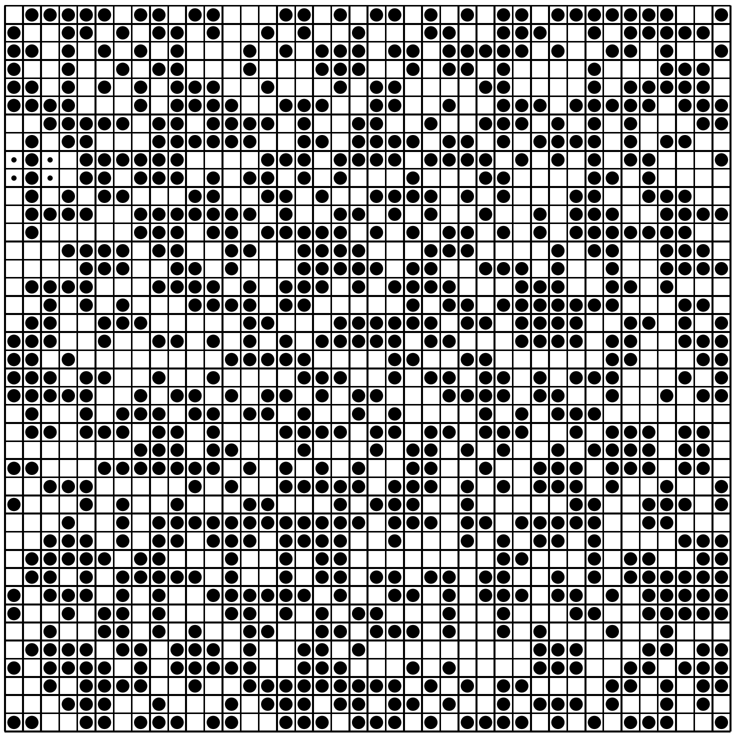


Fig. 6. Randomly generated partially solved 40×40 Nonogram, with 881 black pixels; the 4 small dots denote a pure switching component, leading to 2 solutions

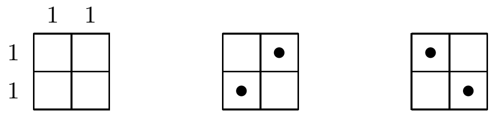


Fig. 7. Pure 2×2 switching component, with its 2 solutions

7 Conclusions

The general Nonogram problem is known to be NP-hard. However, it appears that in practice many instances can be solved quickly. In this paper we presented a general framework for solving Nonograms. By combining several relaxations, that can each be solved in polynomial time, a solution of the Nonogram is computed iteratively. The different solution methods are combined using a 2-SAT formulation. We demonstrated that this approach can solve a variety of interesting Nonograms. More importantly, the algorithm generates a logical proof for all

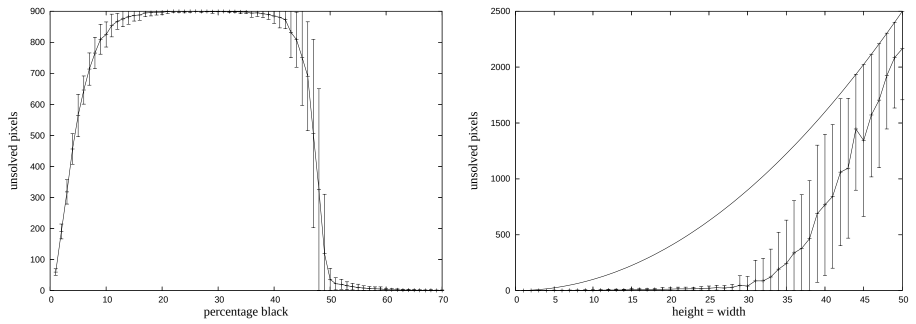


Fig. 8. a) Average number of unsolved pixels with standard deviation, for randomly generated 30×30 puzzles, with different percentages of black pixels; b) idem, for different sizes, with a fixed percentage of 50 %

pixels that are decided. Even if the puzzle cannot be solved completely, it may still be still be possible to decide the value of a substantial part of the pixels. The class of Nonograms that can be solved effectively using our approach includes the simple puzzles that can be found in puzzle books, but also includes random puzzles, which can often not be solved by simple logic reasoning, considering one line at a time.

Our framework is quite general. For example, as indicated in Section 2, the concept of a *description* can be generalized in a straightforward manner. In future work, we intend to study several such generalizations.

References

1. Aharoni, R., Herman, G., Kuba, A.: Binary vectors partially determined by linear equation systems. Discrete Math. **171** (1997) 1–16
2. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Reconstructing convex polyominoes from horizontal and vertical projections. Theoret. Comp. Sci. **155** (1996) 321–347
3. Batenburg, K.J.: A network flow algorithm for reconstructing binary images from discrete X-rays. J. Math. Imaging Vision **27(2)** (2007) 175–191
4. Batenburg, K.J., Kosters, W.A.: A discrete tomography approach to Japanese puzzles. In: Proceedings of the 16th Belgium-Netherlands Conference on Artificial Intelligence, BNAIC (2004) 243–250
5. Garey, M., Johnson, D.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
6. Kuba, A., Balogh, E.: Reconstruction of convex 2D discrete sets in polynomial time. Theoret. Comp. Sci. **283(1)** (2002) 223–242
7. Simpson, S.: Nonogram solver, websites. URL: `www.comp.lancs.ac.uk/~ss/nonogram/links.html` (2007)
8. Ueda, N., Nagao, T.: NP-completeness results for nonogram via parsimonious reductions, preprint. URL: `citeseer.ist.psu.edu/ueda96npcompleteness.html` (1996)
9. Woeginger, G.: The reconstruction of polyominoes from their orthogonal projections. Inform. Process. Lett. **77** (2001) 225–229