Solving Nonograms by combining relaxations

K.J. Batenburg [a] W.A. Kosters [b]

[a] *Vision Lab, Department of Physics, University of Antwerp*
Universiteitsplein 1, B-2610 Wilrijk, Belgium
joost.batenburg@ua.ac.be

[b] *Leiden Institute of Advanced Computer Science, University of Leiden*
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
kosters@liacs.nl

Abstract

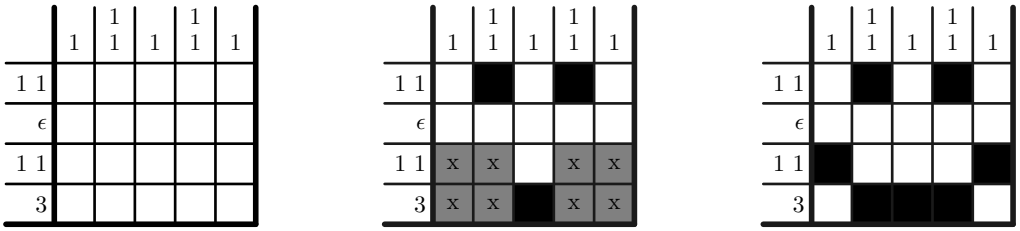
Nonograms, also known as Japanese puzzles, are a specific type of logic drawing puzzles. The challenge is to fill a grid with black and white pixels in such a way that a given description for each row and column, indicating the lengths of consecutive segments of black pixels, is adhered to. Although 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 reasoning framework that can be used to determine the value of certain pixels in the puzzle, given a partial filling. Constraints obtained from relaxations of the Nonogram problem are combined into a 2-SAT problem, which is used to deduce pixel values in the Nonogram solution. By iterating this procedure, starting from an empty grid, it is often possible to solve the puzzle completely. All the computations involved in the solution process can be performed in polynomial time. 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. In addition, we present statistical results on the solvability of Nonograms, obtained by applying our method to a large number of Nonograms.

Key words: Nonograms; Discrete Tomography; Logic reasoning; 2-SAT

1 Introduction

Logic puzzles are very popular nowadays. The most famous example of such a puzzle is the Sudoku, where the puzzler has to insert a series of numbers in a grid, while satisfying certain constraints on the placement of those numbers. A *Nonogram*, also known as a *Japanese puzzle* in some countries, is a different 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. Nonograms require (contrary to Sudoku) some elementary knowledge of integer calculations, so perhaps they cannot be considered as pure logic puzzles. The combination of a logic problem with integer calculations results in a combinatorial problem that can be approached using methods from combinatorial optimization, logical reasoning or both, which makes Nonograms highly suitable for educational use in Computer Science [1].



(a) Original puzzle (b) Partial solution (c) Final solution

Fig. 1. An elementary 5×5 Nonogram. In the partial solution (b), the grey cells indicate which puzzle entries are still undecided. The partial solution cannot be extended to a full solution by using only information from a single row or column at each step.

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 lengths of the consecutive segments of black pixels along the corresponding line, in order. 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, but not fully; see Fig. 1(b). For instance, one can infer that the middle pixel in the bottom row must be black. Using 2-Satisfiability (2-SAT) rules this elementary puzzle can be solved completely; see Fig. 1(c). More complicated puzzles require more sophisticated techniques, as we will also demonstrate.

Nonograms can be considered as a generalization of a well-known problem in Discrete Tomography [2,3]: 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 [4,5], whereas the reconstruction problem for general hv-convex sets is NP-hard [6]. Therefore, the reconstruction problem for Nonograms is also NP-hard (and, clearly, NP-complete). In [7] this is shown through the more general concept of parsimonious reductions. In [8], the general Discrete Tomography problem (without geometric constraints) was analyzed as a simplification of the Nonogram problem. The methods from this paper perform well in this situation, see Section 7.2.

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 many scheduling problems, the type of constraints that occur in Nonograms only apply to the rows, or the columns, but usually not both. The reasoning framework presented in this paper can be extended to include such cases as well. Even for the Nonogram problem there are concrete instances, such as the following. The rows correspond with employees, that have to perform confidential tasks. The employees are ordered, where employees with higher row index perform better. Each employee has a series of work shifts, separated by mandatory breaks. The columns correspond with the consecutive time slots. For each time slot the description defines the numbers of employees needed for the different tasks, where the tasks are ordered in increasing difficulty (needing better employees for more difficult tasks). The confidentiality of the tasks requires that the groups of cooperating employees are physically separated. As an example, in the problem from Fig. 1 all tasks are for single employees, and the best employee has a single shift of 3 time slots.

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. In fact, knowledge about uniqueness of the solution of a Nonogram provides extra information. As an example, for the Nonogram in Fig. 1, a symmetry argument infers the bottom line. On the other hand, large random puzzles can be very difficult to solve, even using a computer, and may have many different solutions — or none at all. 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 [9] for a list of solvers), we have not found any thorough studies of this problem in the scientific literature. In [10], 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. A heuristic algorithm for solving Nonograms is proposed in

[11]. The related problem of *generating* Nonograms that are uniquely solvable is discussed in [12].

Any Nonogram problem can be considered as a *Constraint Satisfaction Problem* (CSP, see, e.g., Chapter 5 from [13]). The line descriptions can be translated into constraints in several different ways. One such method will be discussed later. An advantage of this approach is that a general CSP solver can be used to solve the problem. A wide range of such solvers is available. Formulating a Nonogram problem as a CSP requires a relatively large number of constraints, and its solution lacks the intuitive interpretation characteristic for more specialized Nonogram solvers, such as the approach presented here.

In this paper we propose a reasoning framework for solving Nonograms. We consider relaxations of the Nonogram problem that *can* be solved in polynomial time and, in many cases, can already determine the value of certain pixels in the Nonogram solution. In addition, relations between pairs of pixels are collected from these relaxations, and combined into a 2-Satisfiability (2-SAT) problem. This 2-SAT problem, which may involve information from several rows and columns, can be used to deduce the value of pixels that were not determined directly by the individual relaxations. 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 no further information can be deduced by our solver.

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. In Section 6, (non-)uniqueness properties of Nonograms are discussed. Experimental results are presented in Section 7; Section 8 provides a discussion of advantages and limitations of the proposed approach, and concludes this paper. We note that this paper is an extended version of the conference paper [14].

2 Notation and concepts

We now define notation for a single line (i.e., row or column) of a Nonogram. After that, we combine these into rectangular puzzles.

Let Σ be a finite alphabet. Its elements are referred to as *pixel values*. In this paper we focus on the case $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, but most concepts apply to sets

consisting of more than two elements as well. The symbols `0` and `1` represent the white (`0`) and black (`1`) pixels in the puzzle. In addition, we introduce a special symbol, $\mathtt{x} \notin \Sigma$, indicating that a pixel is not decided yet. Put $\Gamma = \Sigma \cup \{\mathtt{x}\}$. 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 $a_j \leq b_j$ $(j = 1, 2, \ldots, k)$. The 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. Any such d_j will correspond with between a_j and b_j characters σ_j, as defined below. Without loss of generality we will assume that consecutive characters σ_j differ, so $\sigma_j \neq \sigma_{j+1}$ for $j = 1, 2, \ldots, k-1$. 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*. 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. We use 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 $s = \sigma_1^{c_1}\sigma_2^{c_2}\ldots\sigma_k^{c_k}$, where $a_j \leq c_j \leq b_j$ for $j = 1, \ldots, k$. As an example, consider the following description for $\Sigma = \{\mathtt{0}, \mathtt{1}\}$:

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

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

$$0^*1^{a_1}0^+1^{a_2}0^+\ldots1^{a_r}0^*.$$

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

Let s be a finite string over Γ. If zero or more occurrences of `x` 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 *fix*. If a string s has a fix that adheres to a given description d, s is called *fixable with respect to* d. By definition, the boolean function $Fix(s, d)$ is `true` if and only if s is fixable with respect to d. In a somewhat different

context, we also use the term *fixing a pixel* to indicate that a pixel has only one possible value, and can therefore be assigned that value.

An $m \times n$ *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 an $m \times n$ matrix over Γ. The set of all partial fillings is denoted by $\Gamma^{m \times n}$; its elements can also be considered as strings of length $m \times n$. If a partial filling contains no occurrences of $\texttt{x}$, it is called a *full fix*. A full fix $F \in \Sigma^{m \times n}$ *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* and *fixable* that were defined for single lines in the natural way to $m \times n$ Nonograms.

3 Partial solution methods

In this section we study two relaxations of the Nonogram 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.

Although in some cases these relaxations can directly yield information on the value of a given pixel in any solution of the Nonogram, they can also be used to obtain information about *pairs* of pixels. In Section 4, we will show how these relations between pairs of pixels obtained from the relaxed problems can be combined efficiently, obtaining a powerful Nonogram solver.

3.1 Solving a single line

Deciding fixability for a single horizontal or vertical line is a fundamental operation in our approach. Although the number of possible specifications of a line increases exponentially with the number of undecided pixels on that line, it is possible to determine whether or not a line is fixable with respect to its description in polynomial time.

First we introduce some notations. For a string $s = s_1 s_2 \ldots s_\ell$ of length ℓ over Γ, 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 $d = (d_1, d_2, \ldots, d_k)$, 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 $\sigma \in \Sigma$, $s \in \Gamma^\ell$ and $1 \leq i \leq \ell$, let $L_i^\sigma(s)$ denote the largest index $h \leq i$ such that $s_h \notin \{\sigma, \mathtt{x}\}$, if such an index exists, and 0 otherwise. We will put $Fix(i,j) = Fix(s^{(i)}, d^{(j)})$. The value $Fix(\ell, k)$ then determines whether s is fixable with respect to d.

The value $Fix(i,j)$ can be expressed recursively using only terms $Fix(i',j')$ with $i' < i$ and $j' < j$. This allows for efficient evaluation of $Fix(i,j)$ by dynamic programming. As boundary values we note that $Fix(0,j) = \mathtt{true}$ if and only if $A_j = 0$ ($j = 0, 1, 2, \ldots, k$); and $Fix(i,0) = \mathtt{false}$ for $i = 1, 2, \ldots, \ell$. We clearly have $Fix(i,j) = \mathtt{false}$ 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:

Proposition 1 *The function Fix satisfies*

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

This holds for i and j with $1 \leq i \leq \ell$, $1 \leq j \leq k$ and $A_j \leq i \leq B_j$.

Note that an empty disjunction is $\mathtt{false}$; this happens for example if $L_i^{\sigma_j}(s) \geq i - a_j + 1$. For $j = 1$ we have $Fix(i,1) = \mathtt{true}$ if and only if $L_i^{\sigma_1}(s) = 0$.

Proof 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 characters σ_j; say we want σ_j at positions $p+1, p+2, \ldots, i$. We then must have $a_j \leq i - p \leq b_j$. Also note that all elements $s_{p+1}, s_{p+2}, \ldots, s_i$ must be either $\mathtt{x}$ or σ_j; this holds exactly if $L_i^{\sigma_j}(s) \leq p$. 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 terms can be considered to represent general tomographic restrictions. It is natural to implement this recursive formula by means of dynamic programming, using lazy evaluation: once a $\mathtt{true}$ $Fix(p, j-1)$ 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 $\mathtt{x}$ 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 $\mathtt{true}$), 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 $s = s_1 s_2 s_3 s_4 s_5$ 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 $s' = \mathit{Settle}(s, d)$. The complexity of the computation of $\mathit{Fix}(\ell, k)$ is bounded by $k \cdot \ell^2$: at most $k \cdot \ell$ values of $\mathit{Fix}(i, j)$ must be computed, and each such computation can be performed in $O(\ell)$ time, including the evaluation of $L_i^{\sigma_j}(s)$. In practice, especially when using lazy evaluation, the complexity is much lower.

If $|\Sigma| > 2$, it could be that besides deciding certain pixels as in the example from the previous paragraph, also one or more (but perhaps not $|\Sigma| - 1$) pixel values are excluded for certain pixels. In this case, *Settle* can be implemented such that it yields a list of these forbidden values for each pixel. Formulating the Nonogram problem as a *Constraint Satisfaction Problem* (CSP), for each pixel a domain is initialized to Γ and gradually reduced by a search algorithm until only a single value remains.

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 [15] gives a constructive procedure for finding all such pixels.

Fixability to a solution of the DT problem can easily be checked using network flow methods. We refer to [16] for the details of this model. Fig. 2a shows a small 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. This network has three source nodes, one for each column, and three sink nodes, one for each row. The surplus/demand for each node is determined by its corresponding linesum. 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.

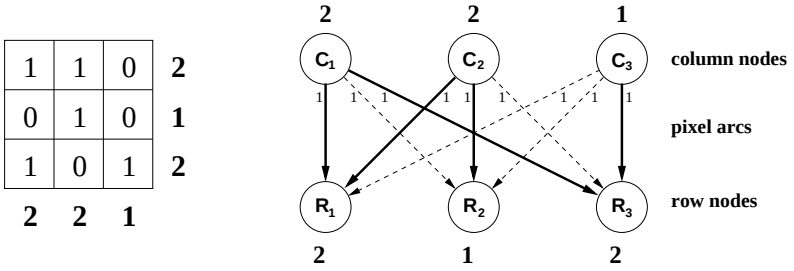


Fig. 2. a) DT problem and one of its solutions, where bold figures denote the linesums; b) associated network of the DT problem

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 a large part of 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 [17]. 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 series of such clauses is simultaneously satisfied. It can be solved in polynomial time, using the concept of a *dependency graph*, as shown in Fig. 3b for our elementary example.

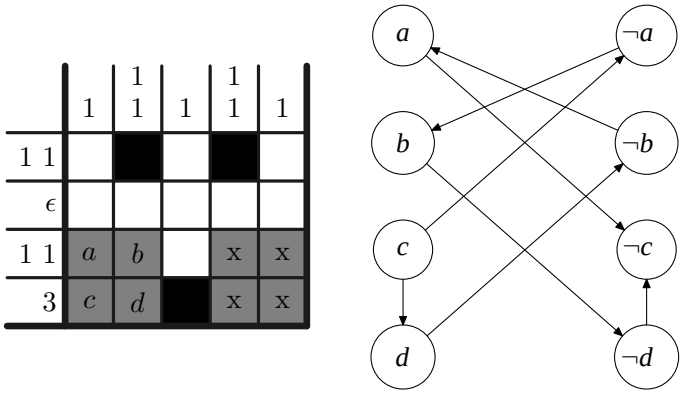


Fig. 3. a) Partially solved Nonogram; b) (part of) its corresponding dependency graph

Combining partial information from different relaxations into a 2-SAT problem allows for deducing substantially more pixel values than can be found using the individual relaxations, even though it does not capture the full puzzle specification. 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.

Note that existence of a cycle in the dependency graph, containing both x and $\neg x$ implies that no satisfying truth assignment exists. If we can assume that a given Nonogram has at least one solution, and that we only fix the value of pixels that must have the same value in all solutions, such a cycle will never occur.

The dependency graph model 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. If x

and y are on one line, for each assignment the fixability computation from Section 3.1 is performed. Each such test that returns `false` provides an additional 2-SAT clause (e.g., $x \vee \neg y$). Similarly, the technique from Section 3.2 can generate clauses, also for pixels that are not on the same line. 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 $\{\mathtt{0}, \mathtt{1}, \mathtt{x}\}$ 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. Although the general 3-SAT problem is NP-hard, there are certain subclasses of 3-SAT problems that *can* be solved in polynomial time [18]. For our purpose, we found the 2-SAT formulation to be both powerful and computationally efficient, yet it may be possible to create a more powerful solver by using a 3-SAT formulation that fits the type of 3-SAT clauses obtained from the relaxed problems and can still be solved in polynomial time.

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 clarity, we now focus on the iterative application of the *Settle* operation from Section 3.1; this yields a powerful approach, as will be shown in Section 7. It can even be combined with other relaxations, as that from Section 3.2, 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, but the order does not affect the result obtained when no further elements can be fixed. The operation of repeatedly applying the *Settle* operation until no further elements are fixed 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 (partial) solution satisfies $Fix(s, d) = \texttt{false}$, there is no full solution.

Now, given $X \in \Gamma^{m \times n}$ and a Nonogram description N such that $X = FullSettle(X, N)$, 2-SAT expressions are collected: for each relaxed problem (e.g., the single rows and columns), for each pair of undecided pixels and for each assignment of values to these pixels, it is checked if for this value assignment the relaxed problem can still be solved. If this is not the case, the relation between both pixel values is captured in a 2-SAT clause. We denote the set of all 2-SAT clauses found in this way by S. Subsequently, all pixels that have the same value in all solutions of the 2-SAT problem S are fixed to these values, as described in Section 4. This operation is called *2SATSolve*: $X \leftarrow$ *2SATSolve*(X, S, N). The operations *FullSettle* and *2SATSolve* can be intertwined, until no further progress is made; this combination is called *Solver0*: $X \leftarrow$ *Solver0*(X, N), cf. Fig. 4. 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 $X =$ *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 $Y_{ij} = \texttt{0}$ and $Y_{ij} = \texttt{1}$. If for one of these *Solver0*(Y, N) contains a contradiction, we know that X_{ij} must have the other value. We can, again in some order, examine all unknown pixels. Note that only those pixels that occur in a 2-SAT clause need to be examined and that the ordering of those pixels does not affect the final results of the solver. This procedure can be repeated, until no further progress is made, again intertwined with the use of *Solver0*. This procedure is called *Solver1*, cf. Fig. 4. If a Nonogram can be solved in this way, it is called 1–*Solvable*.

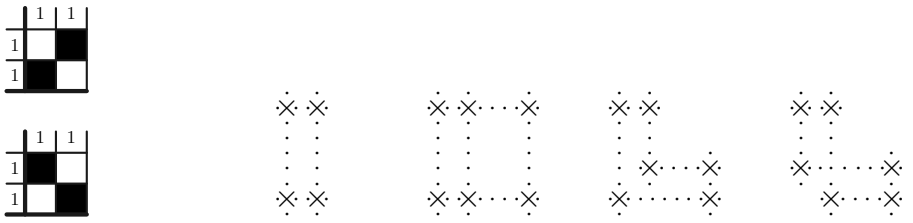
Although the procedure *Solver1* can be considered as a form of *backtracking*, the algorithm only makes a single guess at a time (i.e., no recursive guesses), thereby avoiding the creation of a search tree.

So, 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*. In Section 7, we will give several examples. Also note that *FullSettle*, *Solver0* and *Solver1* are capable of providing partial solutions, where the fixed pixels that have been assigned a value from Σ must have that value in all solutions (if any) of the Nonogram.

Solver0(X, N)
begin
  Input: Nonogram N of size $m \times n$; partial filling X;
  repeat
    $X \leftarrow$ *FullSettle*(X, N);
    Collect a set S of 2-SAT clauses from relaxations;
    $X \leftarrow$ *2SATSolve*(X, S, N);
  until (no new fixed pixel(s) found) **or** (contradiction found);
  if (contradiction found) Output: "contradiction";
  else Output: X;
end

Solver1(X, N)
begin
  Input: Nonogram N of size $m \times n$; partial filling X;
  repeat
    $X \leftarrow$ *Solver0*(X, N);
    repeat
      Select an undecided pixel X_{ij};
      for each pixel value $\sigma \in \{0, 1\}$
        $Y \leftarrow X$; $Y_{ij} \leftarrow \sigma$;
        $Y \leftarrow$ *Solver0*(Y, N);
        if (contradiction found) $X_{ij} \leftarrow 1 - \sigma$;
    until (new pixel value(s) found) **or** (all undecided pixels tried);
  until (no new fixed pixel(s) found);
  Output: X;
end

Fig. 4. High level overview of the *Solver0* and *Solver1* algorithms.



(a) Elementary switching component

(b) More complex types of switching components for Nonograms

Fig. 5. Switching components that can occur in Nonograms, containing up to six unknowns.

6 Some remarks on (non-)uniqueness

In this section, we examine some issues concerning (non-)solvability of Nonograms, and we pay special attention to puzzles with multiple solutions.

Similar to the Discrete Tomography problem, Nonograms can have more than one solution. However, where the non-uniqueness problem for Discrete Tomography allows an elegant description based on elementary switching components [19], the non-uniqueness problem for Nonograms appears to be much more complex. The problem of deciding if another solution of a Nonogram exists, given a particular solution, is NP-hard [7].

We now focus on cases, where only a small subset of the pixel values is not uniquely determined by the Nonogram description. Suppose that we have a given Nonogram, and that it is possible (using the approach of this paper, for example) to determine the value of all pixels, except for a small set of $u > 0$ unknown pixels or *unknowns*. We first note that each line with unknown pixels should contain at least 2 unknowns. This implies that $u \geq 4$ and $u \neq 5$. If $u = 4$, the unknowns form a rectangle (Fig. 5b, left), similar to the elementary *switching component* of the Discrete Tomography problem (i.e., four pixels, two black and two white, such that interchanging the black and the white pixels does not change the description; see Fig. 5a). However, in Nonograms the existence of such switching components is not only determined by the value of the four corner pixels, but also by the values of the pixels along the four sides of the rectangle and by the pixels adjacent to the four corners. On a single line with 2 unknowns, depicted from left to right, we note that left of the leftmost unknown we must have a `0` pixel (or the image boundary), and a similar observation holds for the rightmost unknown. We note that we have precisely 2 solutions here. Between the two unknowns we must have only `1`s, or a series of solitary `1`s with variable length blocks of `0`s in between: in regular expression notation $(\texttt{0}^+\texttt{1})^*\texttt{0}^+$.

If we consider $u = 6$, we either have two lines with 3 unknowns each (and in the other direction three lines with 2 unknowns each), or three lines with 2 unknowns each in both directions (Fig. 5, right). In the former case we have 3 solutions, where in between two unknowns we can only have the $(\texttt{0}^+\texttt{1})^*\texttt{0}^+$ situation, in the latter case there are precisely 2 solutions — as in the $u = 4$ case. In all these situations, unknowns cannot touch.

7 Experimental results

In this section, we present a variety of experimental results obtained with the proposed iterative Nonogram solvers. All considered puzzles will have at least one solution: the image that was used to construct the puzzle. We mention that in our experience, most puzzles from newspapers are of the 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. On the other hand, the 80×50 puzzle shown in Fig. 6 is rather large, but can still be solved by hand, belonging to the simple type. Nevertheless, the larger the puzzle, the more complicated it can be. All puzzles of simple type can be solved very fast by the *FullSettle* operation, as the dynamic programming approach described in Section 3.1 effectively captures all information contained in the description of each single horizontal or vertical line.

This section is structured as follows. First, we consider random images, presenting both characteristic examples and statistical results. Next, we consider the case of hv-convex images. Finally, we provide solvability results for the set of *all* Nonograms of size 5×5.

Unless mentioned otherwise, the experimental results have been obtained by *Solver1*, which is the most powerful method of those described in this paper. Our focus is on the solution quality, determined by the number of unknowns left after running the Nonogram solver. For the hv-convex images in Section 7.2, we also present timing results. For all experimental results presented, the running time for a single Nonogram never exceeded 60 seconds (and is usually much shorter) on a Pentium IV PC, running at 2.4 GHz. For general images, there is no direct relationship between the reconstruction time and the size of the Nonogram, as the reconstruction time is largely determined by the complexity properties of the particular image being reconstructed.

7.1 Random Nonograms

As an illustrative example for a more difficult puzzle, consider the 30×30 Nonogram in Fig. 7a. The Nonogram was randomly generated with 50 % black pixels. Using only *FullSettle* just 11 pixel values can be determined. Using *Solver1* (which only takes approximately one second on a standard 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 can easily be accomplished by repeatedly fixing one of the unknowns to either `0` or `1`, and then re-solving the puzzle. This Nonogram

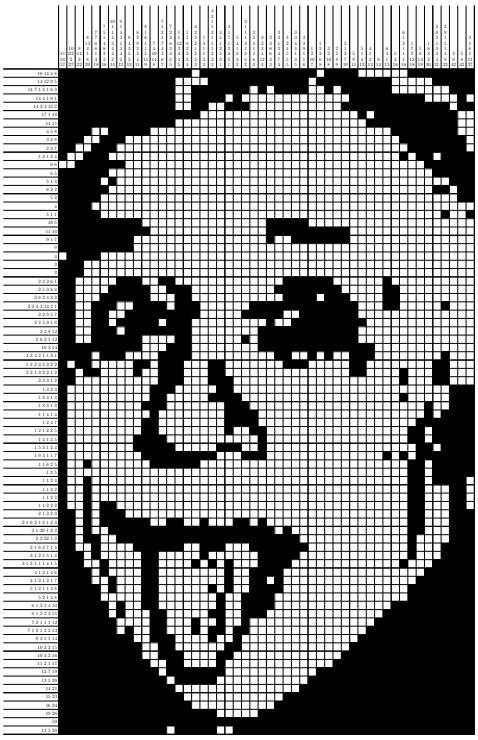
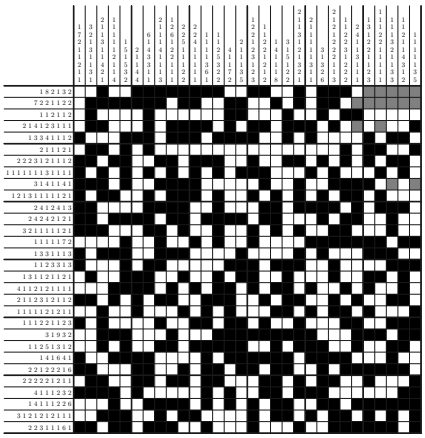
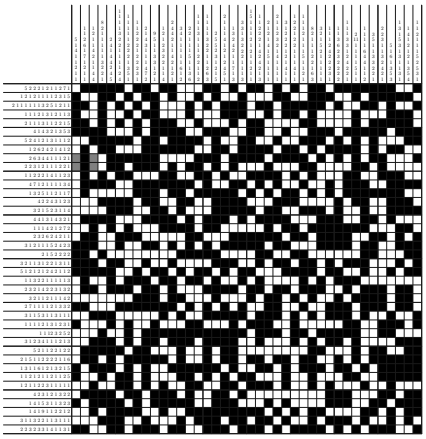


Fig. 6. Large Nonogram of the simple type



(a) Random 30×30



(b) Random 40×40

Fig. 7. a) Randomly generated partially solved 30×30 Nonogram, with 50 % black pixels; the grey cells denote the unknown pixels. This Nonogram has six solutions; b) Randomly generated partially solved 40×40 Nonogram, with 881 black pixels; the four grey pixels denote an elementary switching component, leading to two solutions.

was also included in [10], 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 current approach finds the set of all solutions (or rather those pixels that have the same value in every solution), 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. In particular, pixels that can be either black or white, depending on the particular solution of the Nonogram, can never be fixed in our approach.

Fig. 7b shows 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 elementary 2×2 switching component (cf. Fig. 5a), so there are 2 different solutions. Again, *Solver1* is necessary: *FullSettle* finds 101 pixels.

In Fig. 8a, results for a set of 7,000 images are summarized. For each p in $\{1, 2, \ldots, 70\}$ the *Solver1* 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 for each point on the graph, truncated at 0 and 900. Fig. 8b shows the results of applying the three methods *FullSettle*, *Solver0* and *Solver1* to these puzzles (the graph for *Solver1* being the same as that from Fig. 8a). Note that between 40 % and 60 % of black pixels, the behaviour of the three algorithms clearly differs, where indeed *Solver1* performs best.

For small and large black percentages nearly all puzzles are solvable, in some cases leaving small switching components. Fig. 9 shows, for each size s in $\{1, 2, 3, \ldots, 50\}$, the average number of unsolved pixels for randomly generated square $s \times s$ Nonograms, all with 50 % black pixels. The results are averaged over 100 runs for each graph point. The smooth curve in Fig. 9a depicts the total number of pixels, i.e., s^2. Fig. 9b shows percentages.

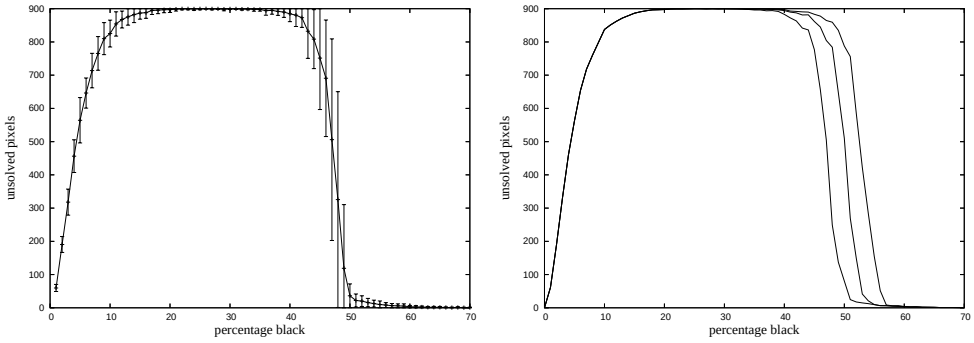


Fig. 8. a) Average number of unsolved pixels for randomly generated 30×30 puzzles, for a varying percentage of black pixels, when using *Solver1*; error bars indicate the standard deviation. b) As a), for *FullSettle* (top graph), *Solver0* (middle graph) and *Solver1* (bottom graph), without standard deviation

Fig. 10 depicts, for each size s in $\{10, 11, 12, \ldots, 50\}$, the average percentage of unsolved pixels over 20 runs, for randomly generated square $s \times s$ puzzles, where the percentage of black pixels ranges from 40 % to 60 %. This percentage

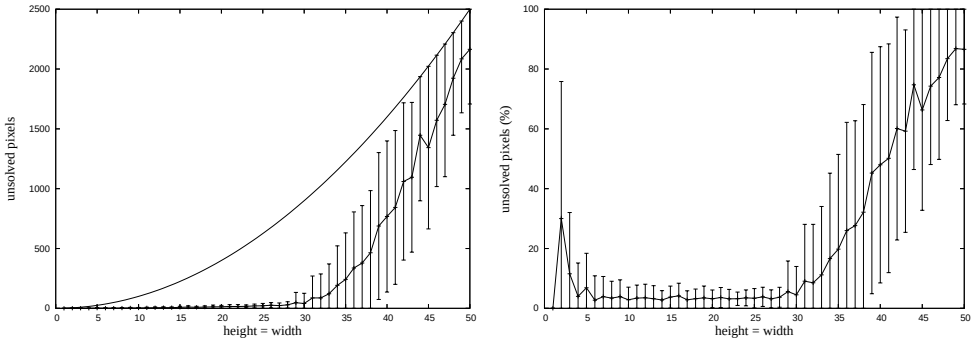


Fig. 9. a) Average number of unsolved pixels for randomly generated puzzles of different size, with a fixed percentage of 50 % black pixels; error bars indicate the standard deviation. b) As a), but now showing these values as percentage of the total number of pixels

range has been chosen to highlight the transition from completely unsolvable puzzles to largely solvable ones (cf. also Fig. 8b). Here, we compare the results for the less powerful *FullSettle* solver that only uses the information from single lines (Fig. 10a) with *Solver1* (Fig. 10b). Even though the general shape of both graphs is similar, it is clear that the 2-SAT approach employed by *Solver1* results in a significantly larger number of solved pixels.

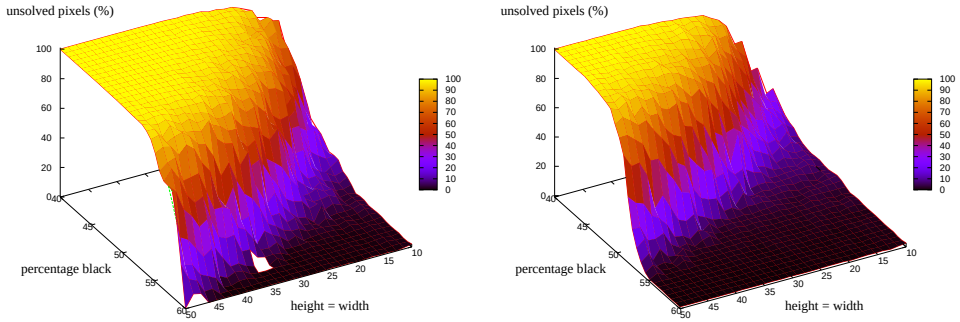


Fig. 10. a) Average percentage of unsolved pixels, for randomly generated puzzles of different size and black percentage, for *FullSettle*. b) Idem, for *Solver1*

7.2 hv-convex images

The class of *hv-convex* images forms an interesting test case for Nonogram solvers. For such images, the Nonogram problem is identical to the problem of reconstructing hv-convex images from given linesums, which has been studied extensively in the Discrete Tomography literature.

The framework for generating certain discrete images using uniform distributions proposed in [20] was utilized to generate a set of hv-convex test images. In preliminary experiments, it was found that if the test images are sampled from the set of *all* hv-convex images, they typically contain a large number

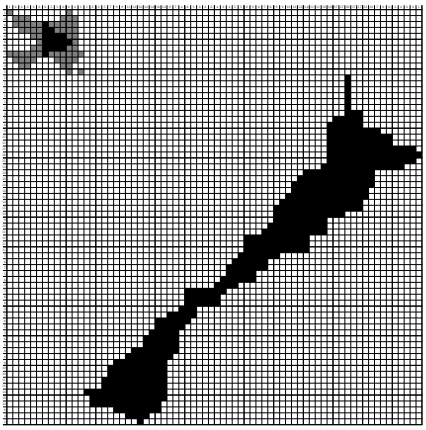
components size	1	2	3	4	avg. time (s)
10×10	1.53	4.10	13.57	36.30	0.03
20×20	0.72	2.13	7.15	16.40	0.1
30×30	0.35	2.53	4.63	9.81	0.4
40×40	0.32	1.23	4.06	5.87	1.5
50×50	0.29	1.29	2.79	5.77	2.7
60×60	0.46	1.50	2.42	4.81	7.1
70×70	0.27	1.29	2.88	3.68	15.5
80×80	0.56	0.81	1.74	3.49	25.4

Table 1
Reconstruction results for the *Solver1* algorithm for the dataset of hv-convex images. For each test case, the average number of unknown pixels is shown.

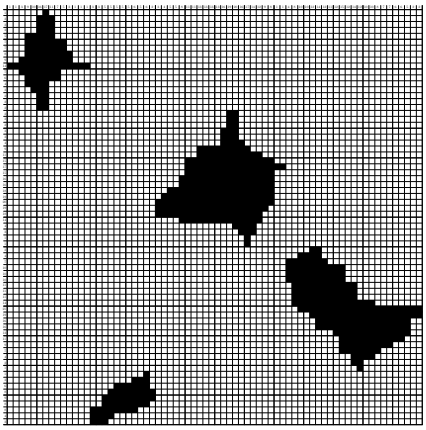
of very small hv-convex components. Such Nonograms often have a very high degree of non-uniqueness, which makes them unattractive as test images for our algorithms. We therefore restricted the test set to hv-convex images that have a fixed number of components, ranging from 1 up to 4. We only considered images that contain at least one black pixel in each row and column. For each number of components, and for sizes ranging from 10×10 up to 80×80, a set of 100 images was generated, sampled uniformly from the collection of all hv-convex images having that size and number of components.

Table 1 shows the average number of unknown pixels left by the *Solver1* algorithm for each test case. The rightmost column shows the average running time in seconds over all test cases of each size. We can observe two general patterns in the reconstruction results. Firstly, the number of unknown pixels increases with the number of components in the image. Secondly, the number of unknown pixels generally decreases with the image size.

As the number of components increases, the number of solutions of the Nonogram will typically also increase. In fact, for the limiting case where the number of components equals the number of rows (and columns), any permutation matrix will be a solution. As our algorithm will only determine the value of pixels that have the same value in all solutions, this non-uniqueness directly leads to unknown pixels in the reconstruction. A second explanation for the increase of unknown pixels with the number of components, is that the average number of black pixels on each line will decrease as the number of components increases. Long segments of 1's are very useful in solving Nonograms, as it is usually possible to directly determine a subset of pixels that must be 1 in any solution. An example of the non-uniqueness problem is shown in Fig. 11a. The hv-convex component in the top-left corner cannot be determined, even



(a) Nonogram that was not solved completely



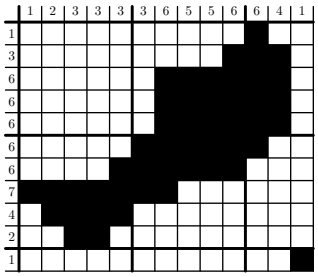
(b) Nonogram that was solved completely

Fig. 11. Two 70×70 Nonograms from the dataset of hv-convex images.

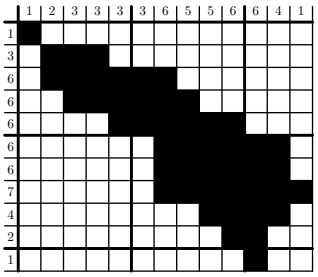
though there is a large number of 2-SAT relations in that region. Further inspection reveals that there are in fact two solutions for this region, shown in Fig. 12. In such cases, combining our reasoning framework with a simple branching algorithm would easily find both solutions. In cases where there is no such non-uniqueness, we found that our reasoning approach is nearly always capable of completely solving the Nonogram. An example image of size 70×70 consisting of four hv-convex components that is solved completely, is shown in Fig. 11b.

Even though the number of unknown pixels generally decreases with the size of the image, the columns of Table 1 are not strictly decreasing, because of outliers. For example, the 70×70 image in Fig. 11a with 3 components, leads to 62 unknowns with a contribution of 0.62 to the table value.

The running time clearly increases more than linearly with the number of pixels in the image. Still, rather large Nonograms containing thousands of unknowns can be solved within one minute.



(a) First solution



(b) Second solution

Fig. 12. Two hv-convex images corresponding to the Nonogram description in the top-left corner of the larger hv-convex Nonogram in Fig. 11.

Consider Nonogram descriptions of size $n \times n$ for integer $n > 1$. If we start with any of the 2^{n^2} possible positions, say S, we can see whether *FullSettle* is capable of solving the corresponding puzzle, i.e., retrieving S; in this case the puzzle is clearly uniquely solvable. We can also verify whether the stronger *Solver1* can solve the puzzle. For the case $n = 5$, we have reconstructed all possible Nonograms using both solvers. Fig. 13 shows the number of positions leaving u unknowns, for all $2^{5^2} = 2^{25} = 33{,}554{,}432$ possible positions, for these two solvers, where $u > 5$. The dashed line is for *Solver1*, the solid line for *FullSettle*. Note that $u = 1, 2, 3, 5$ cannot occur, as observed before. For $u = 0$, i.e., the uniquely solvable situations, we have 24,976,511 (*FullSettle*) and 25,309,575 (*Solver1*) positions, respectively. For $u = 4$, these numbers are 4,363,030 and 4,623,570, respectively. These cases amount to 87%, resp. 89%, of all positions. Note that in general (among others) the situations with exactly one black pixel in every line give rise to puzzles with a maximal value for u, i.e., $u = n^2$.

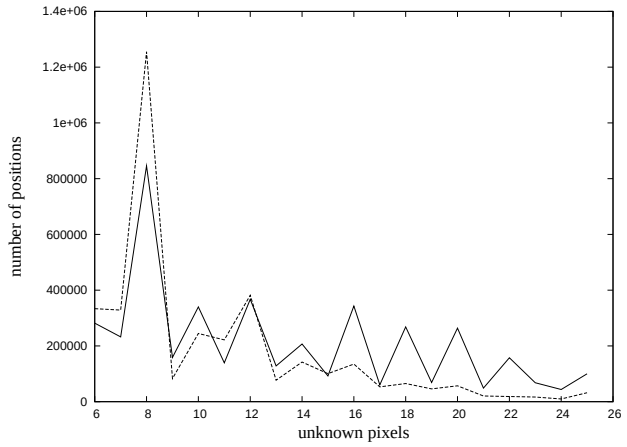


Fig. 13. Number of positions for 5×5 Nonograms with a given number of unknowns after solving, for two different solvers: *FullSettle* (solid line) and *Solver1* (dashed line)

Even if a Nonogram does have a unique solution, our approach of combining 2-SAT rules from several relaxed problems may not capture the full set of constraints present in the Nonogram and, as a consequence, get stuck prematurely without being able to make any further deductions. This can even occur for very small Nonograms, as demonstrated by the example in Fig. 14. Here the six white pixels are easily inferred, while all other pixels (colored grey) are still undecided. In this case, even *Solver1* cannot make any further progress, while one can still infer that the rightmost pixel in the third row must be white. Of course, such small examples could easily be solved by a branching algorithm.

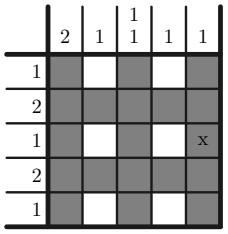


Fig. 14. Partially solved 5×5 Nonogram, where the fact that pixel `x` must be white (`0`) is hard to infer. Note that the grey pixels are still undecided.

8 Discussion and Conclusions

The general Nonogram problem is known to be NP-hard. However, it appears that in practice many instances can be solved quickly. 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 pixels that are decided. Even if the puzzle cannot be solved completely, it may still be possible to decide the value of a substantial part of the pixels. The proposed method does not have to start from scratch: if prior knowledge is available on the value of certain pixels, it can be used to deduce the value of the remaining ones.

The approach is closely related to the reasoning of human puzzlers, in particular for puzzles of the simple type. Even for more complex instances, where information from several lines is combined, our framework yields an intuitive deduction. In this way, the method allows for determining a difficulty level.

Alternatively, as mentioned in the Introduction, Nonograms can be considered as instances of general CSP problems. This approach was, e.g., explored in [21,22], where SAT solvers are employed. Note that a Nonogram puzzle is first translated into a (large) CSP, which is further translated into a SAT problem, and finally solved by a SAT solver. For instance, for the problem from Fig. 7, a SAT problem with 20,548 variables and 31,499 clauses is built. Solution times are comparable with our approach. However, apart from but also caused by the large intermediate problem size, the solution does not give clear clues concerning the deduction, because of the translation steps involved.

Our reasoning framework can be considered as a CSP solver tailored for the specific problem category, which works directly in the original problem domain, without any translation steps involved. In particular for puzzles of the *simple* type, where it is not necessary to harvest 2-SAT expressions and only the dynamic programming approach from Section 3.1 is utilized, huge Nono-

grams can be solved quickly without the need to generate large numbers of constraints as required by a standard CSP formulation.

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 images and hv-convex images, which can often not be solved by simple logic reasoning, considering one line at a time. For the simple puzzles the method is extremely fast, even if the puzzles are very large.

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.

Acknowledgements

The authors are grateful to Péter Balázs for providing an extensive set of hv-convex test images, and to Naoyuki Tamura for supplying his CSP solver. The first author acknowledges the Research Foundation Flanders (FWO) for financial support.

References

[1] S. Salcedo-Sanz, J. Portilla-Figueras, E. Ortiz-Garcia, A. Perez-Bellido, X. Yao, Teaching advanced features of evolutionary algorithms using Japanese puzzles, IEEE Trans. Education 50(2) (2007) 151–156.

[2] G. T. Herman, A. Kuba (Eds.), Discrete Tomography: Foundations, Algorithms and Applications, Birkhäuser, Boston, 1999.

[3] G. T. Herman, A. Kuba (Eds.), Advances in Discrete Tomography and its Applications, Birkhäuser, Boston, 2007.

[4] A. Kuba, E. Balogh, Reconstruction of convex 2D discrete sets in polynomial time, Theoret. Comp. Sci. 283(1) (2002) 223–242.

[5] E. Barcucci, A. Del Lungo, M. Nivat, R. Pinzani, Reconstructing convex polyominoes from horizontal and vertical projections, Theoret. Comp. Sci. 155 (1996) 321–347.

[6] G. Woeginger, The reconstruction of polyominoes from their orthogonal projections, Inform. Process. Lett. 77 (2001) 225–229.

[7] N. Ueda, T. Nagao, NP-completeness results for Nonogram via parsimonious reductions, preprint (1996).
URL `citeseer.ist.psu.edu/ueda96npcompleteness.html`

[8] J. Benton, R. Snow, N. Wallach, A combinatorial problem associated with Nonograms, Linear Algebra Appl. 412(1) (2007) 30–38.

[9] S. Simpson, Website Nonogram solver [accessed 7.7.2008] (2007).
URL www.comp.lancs.ac.uk/~ss/nonogram/links.html

[10] K. J. Batenburg, W. A. Kosters, A discrete tomography approach to Japanese puzzles, in: Proceedings of the 16th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC), 2004, pp. 243–250.

[11] S. Salcedo-Sanz, E. Ortiz-Garcia, A. Perez-Bellido, J. A. Portilla-Figueras, X. Yao, Solving Japanese puzzles with heuristics, in: Proc. IEEE Symposium on computation intelligence and games, Honolulu, USA, 2007, pp. 224–231.

[12] E. Ortiz-Garcia, S. Salcedo-Sanz, J. Leiva-Murillo, A. Perez-Bellido, J. Portilla-Figueras, Automated generation and visualization of picture-logic puzzles, Computers and Graph. 31 (2007) 750–760.

[13] S. Russell, P. Norvig, Artificial Intelligence, A Modern Approach, 2nd Edition, Prentice Hall, 2003.

[14] K. J. Batenburg, W. A. Kosters, A reasoning framework for solving Nonograms, in: Proceedings of the International Workshop on Combinatorial Image Analysis (IWCIA 2008), LNCS 4958, 2008, pp. 372–383.

[15] R. Aharoni, G. Herman, A. Kuba, Binary vectors partially determined by linear equation systems, Discrete Math. 171 (1997) 1–16.

[16] K. J. Batenburg, A network flow algorithm for reconstructing binary images from discrete X-rays, J. Math. Imaging Vision 27(2) (2007) 175–191.

[17] M. Garey, D. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.

[18] D. Cohen, P. Jeavons, M. Gyssens, A unified theory of structural tractability for constraint satisfaction problems, J. Computer and System Sciences 74(5) (2008) 721–743.

[19] H. J. Ryser, Combinatorial properties of matrices of zeros and ones, Canadian Journal of Mathematics 9 (1957) 371–377.

[20] P. Balázs, A framework for generating some discrete sets with disjoint components using uniform distributions, Theoret. Comp. Sci. 406 (2008) 15–23.

[21] N. Tamura, Website Sugar [accessed 21.10.2008] (2008).
URL bach.istc.kobe-u.ac.jp/sugar/

[22] N. Tamura, T. Tanjo, M. Banbara, System description of a SAT-based CSP solver Sugar, in: Proceedings of the Third International CSP Solver Competition, 2008, pp. 71–75.