Nonograms

K. Joost Batenburg* Walter A. Kosters†

Abstract

Nonograms, also known as *Japanese puzzles*, are in fact image reconstruction problems that can be solved by logic reasoning. Nonograms can have widely varying difficulty levels. Although the general Nonogram problem is NP-hard, the instances that occur in puzzle collections can usually be solved by hand.

This paper focuses on a subclass of Nonograms that can be solved by a sequence of local reasoning steps. A *difficulty measure* is defined for this class of so-called *simple* Nonograms, which corresponds to the number of steps required to reconstruct the image. In the first part of this paper, we investigate the difficulty distribution among this class, analyze the structure of Nonograms that have lowest difficulty, and give a construction for the asymptotically most difficult problems. We also provide some graphs to give insight into the search spaces of the problems at hand. The second part of the paper deals with the task of constructing Nonograms, based on a given gray level image. We propose an algorithm that generates a set of Nonograms of varying difficulty that all resemble the input image. Finally we mention some issues for non-simple Nonograms.

1 Introduction

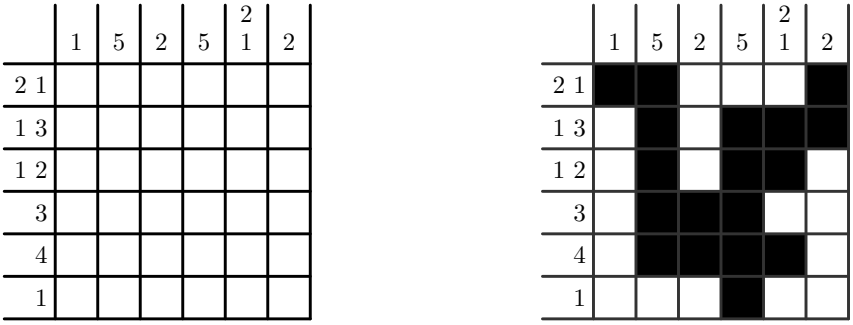
A *Nonogram*, also known as a *Japanese puzzle* in some countries, is a type of logic puzzle which can be considered as an image reconstruction problem. The goal is to find an image on a rectangular pixel grid that adheres to certain row and column (briefly: line) constraints. Usually, the image is black-and-white, although Nonograms with more than two gray values exist as well. In addition to elementary logic, solving Nonograms requires some elementary integer calculations. 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 [13, 16].

From a more general point of view, Nonograms fit into the concept of *Discrete Tomography*. This is an extensive research area, closely related to the field of *Computed Tomography*, with applications ranging from medicine to crystallography. For more information, the interested reader is referred to [6, 7, 4, 5].

Figure 1(a) shows an example of a small Nonogram. Its (unique) solution is shown in Figure 1(b). The *Nonogram description* for each row and column indicates the order and length of consecutive unconnected black segments along those lines. For example, the Nonogram description "2 1" in the first row indicates that from left to right, the row contains a black segment of length 2 followed by a single black pixel. The black segments are separated by one or more white pixels and there may be additional white pixels before the first segment, and after the last segment.

Several implementations of Nonogram solvers can be found on the Internet; see, e.g., [14, 19]. In [2, 10, 15, 16], evolutionary algorithms are described for solving Nonograms; and in [18] neural networks are used. A heuristic algorithm for solving Nonograms is proposed in [12]. The related problem of *constructing* Nonograms that are uniquely solvable is discussed

*CWI, Amsterdam, The Netherlands; K.J.Batenburg@cwi.nl
†LIACS, Universiteit Leiden, The Netherlands; kosters@liacs.nl



(a) 6×6 Nonogram (b) Solved Nonogram

Figure 1: A small Nonogram and its unique solution.

in [9]. In [3], a reasoning framework is proposed for solving Nonograms that uses a 2-SAT model for efficient computation of reasoning steps.

In [17], it was first proved that the general Nonogram problem is NP-hard. This also follows from the fact that Nonograms can be considered as a generalization of the reconstruction problem for hv-convex sets in discrete tomography, which is NP-hard [20]. On the other side of the difficulty spectrum are the Nonograms that can be found in puzzle collections, which can usually be solved by hand, applying a sequence of elementary reasoning steps. In this paper, we focus on this latter class of Nonograms, referred to as the *simple* type in [3]. Such Nonograms can be solved without resorting to branching, yet there can still be a large variance in the number of steps required to find solutions. In [1] a difficulty measure for this class is proposed and analyzed. In particular, a construction for a family of Nonograms that have asymptotically maximal difficulty, up to a constant factor, is provided.

This paper, based on [1, 3], is structured as follows. In Section 2, notation is introduced to describe the objects of this paper and their properties. We provide an efficient algorithm to process single lines, leading to a Nonogram solver that deals with so-called simple Nonograms. Both the simple class and the difficulty measure are defined in Section 3, where also further motivation is provided for studying this particular difficulty measure, and its distribution is analyzed for small Nonograms. Section 4 considers the question what the maximum difficulty can be, as a function of Nonogram size. A construction is given that obtains asymptotically maximal difficulty for Nonograms of arbitrarily large size. Section 5 deals with an application of this difficulty concept: constructing Nonograms of varying difficulty that resemble a gray level input image. An algorithm is proposed for this task, illustrated by some computational experiments. In Section 6 we mention some results (taken from [3]) that extend beyond those for simple Nonograms. Section 7 concludes this paper.

2 Basic notions and algorithms

We first define notation for a single line (i.e., row or column) of a Nonogram. After that, we combine these into rectangular puzzles. Let $\Sigma = \{0, 1\}$, the alphabet of pixel values (more general alphabets are also allowed). We usually refer to 1 as *black* and 0 as *white*. While solving a Nonogram, the value of a pixel can also be *unknown*. Let $\Gamma = \Sigma \cup \{?\} = \{0, 1, ?\}$, where the symbol ? refers to the unknown pixel value.

A *(general) 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 $j = 1, 2, \ldots, k-1$. 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 $a_j \le c_j \le b_j$ for $j = 1, \ldots, k$. As an "example", consider the following description:

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

with $a_i > 0$ $(i = 1, 2, \ldots, r)$. This is exactly what we consider to be a *Nonogram description* $a_1 a_2 \ldots a_r$ for a line (row or column), where we only mention the lengths of consecutive non-touching series of 1s. Note that it has length $2r + 1$ and can also be written as $0^*1^{a_1}0^+1^{a_2}0^+\ldots1^{a_r}0^*$.

A string $s \in \Gamma^\ell$ $(\ell \ge 0)$ can be *(fully) fixed* to a string $t \in \Sigma^\ell$ (referred to as a *fix*) if $s_j = t_j$ whenever $s_j \in \Sigma$ $(1 \le j \le \ell)$. Loosely speaking, one should replace the ?s, or unknowns, with pixel values; we also say that we fix these string elements. If $s \in \Gamma^\ell$ can be fixed to a string in Σ^ℓ that adheres to a given description d, s is called *fixable* with respect to d; in that case the Boolean function value $Fix(s, d)$ is defined to be `true`, and otherwise `false`. The formal operation $Settle\,(s, d)$ constructs a (unique) string from a fixable string s and a description d by replacing all ? symbols in s for which all strings in Σ^ℓ that adhere to the description d have the same unique value, by this value. In other words, all pixels that must have a certain value in order to adhere to the description, are set to that value. As an example, for $s =$?1?1?0????? (with $\ell = 11$) and Nonogram description $d = 3\ 2\ 1$ (so general description $0^*1^30^+1^20^+1^10^*$), we have $Settle\,(s, d) = 011100?1???$.

In [3], an efficient, polynomial-time algorithm is described for performing the *Settle* operation on a string, by using dynamic programming. Before we elaborate on this, we introduce some more general notations, also because it makes it easier to formulate our results. For a string $s = s_1 s_2 \ldots s_\ell$ of length ℓ over Γ, define its prefix of length i by $s^{(i)} = s_1 s_2 \ldots s_i$ $(1 \le i \le \ell)$, 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 $1 \le j \le k$, 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 $1 \le i \le \ell$, let $L_i^\sigma(s)$ denote the largest index $h \le i$ such that $s_h \notin \{\sigma, ?\}$, 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 $Fix(0, j) =$ `true` if and only if $A_j = 0$ $(j = 0, 1, 2, \ldots, k)$; and $Fix(i, 0) =$ `false` for $i = 1, 2, \ldots, \ell$. We clearly have $Fix(i, j) =$ `false` if $i < A_j$ or $i > B_j$ $(0 \le i \le \ell,\ 0 \le j \le k)$, as indicated above.

The main recursion, valid for general alphabets Σ, is:

Proposition 1 *The function Fix satisfies*

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

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 $L_i^{\sigma_1}(s) = 0$.

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 $a_j \le i - p \le b_j$. Also note that all elements $s_{p+1}, s_{p+2}, \ldots, s_i$ must be either ? 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 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 `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 unknowns 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 values 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. (Here the fact that $|\Sigma| = 2$ comes in handy.) Note that for this purpose lazy evaluation is not an option, since we need to examine all fixes.

The complexity of the computation of $Fix(\ell, k)$ is bounded by $k \cdot \ell^2$: at most $k \cdot \ell$ values of $Fix(i, j)$ 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.

An $m \times n$ *Nonogram puzzle description* D consists of $m > 0$ row Nonogram descriptions $r_1, r_2, \ldots, r_m$ and $n > 0$ column Nonogram descriptions $c_1, c_2, \ldots, c_n$. An *image* $P = (P_{ij}) \in \Sigma^{m \times n}$ *adheres* to the description if all lines adhere to their corresponding description. A *Nonogram* N consists of a pair (D, P), where D is a Nonogram puzzle description and P is an image in $\Gamma^{m \times n}$. Usually we will assume that all lines in P are still fixable with respect to their corresponding Nonogram descriptions. Solving such a puzzle means finding an image $P' \in \Sigma^{m \times n}$ that adheres to D, and where every line in P is fixed to the corresponding line in P'. The image P can be viewed as a partial solution.

3 The difficulty of simple Nonograms

A Nonogram puzzle description is called *simple* if it can be (uniquely) solved by applying a sequence of *Settle* operations, each time using only information from a single row or column, starting from an image with only ?s. In other words, it is never necessary to consider information from several rows and columns simultaneously. Nearly all Nonograms that appear in puzzle collections satisfy this property. From this point on, we focus on the class of *simple* Nonograms. Note that for Nonograms of the simple type, there is a bijective map between the set of images and their descriptions. Therefore, we sometimes use the term *Nonogram* to refer to either the image, or its description.

Even though the order of applying the *Settle* operations does not affect whether or not a solution can be found, the required number of operations depends heavily on the order in which rows and columns are selected. We define the following operations:

- The operation *h-sweep* (N) applies the *Settle* operation to all rows of the Nonogram N: a horizontal sweep.
- The operation *v-sweep* (N) applies the *Settle* operation to all columns of the Nonogram N: a vertical sweep.

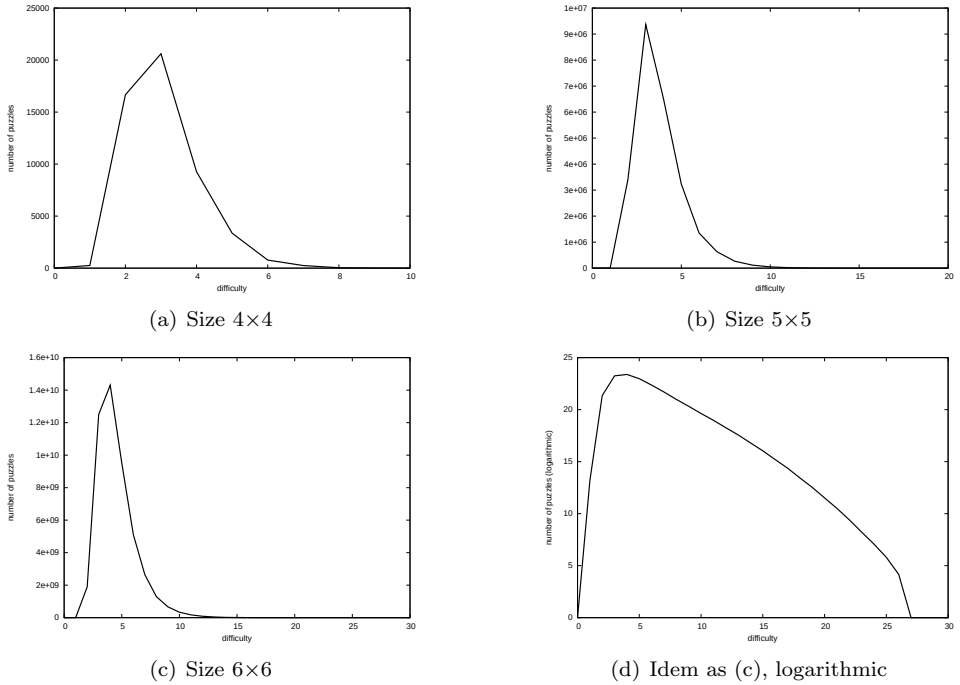
Both operations return the "updated" Nonogram, usually having fewer unknowns.

Now the difficulty of a Nonogram of the simple type is determined by starting with an image N for which all pixel values are unknown, and running the algorithm in Figure 2. The algorithm starts with a horizontal sweep, intertwines horizontal and vertical sweeps, and counts the total number of sweeps until the Nonogram is solved. It is clear that any $m \times n$ Nonogram of simple type has complexity at most equal to $mn+1$, since every sweep (except perhaps the first one) must at least fix one unknown pixel. By definition, the algorithm terminates if and only if N is of the simple type.

Difficulty (N) :
  diff $\leftarrow 0$;
  while N is not solved **do**
    if *diff* is even **then** $N \leftarrow$ *h-sweep* (N);
    else $N \leftarrow$ *v-sweep* (N); **fi**
    diff $\leftarrow$ *diff* $+ 1$;
  od
  return *diff*;

Figure 2: Algorithm that solves a *simple* Nonogram and determines its difficulty.

We remark that our definition of "difficulty" is rather subjective. Quantifying the amount of work required to solve a particular Nonogram is not straightforward, as it depends on the particular solution strategy employed. In Section 5, we will consider the task of constructing Nonograms of *varying difficulty* that resemble a gray level input image. As these Nonograms are intended to be solved by human puzzlers, it is important that the difficulty measure corresponds to the amount of work required by a puzzler to solve the Nonogram. We observed that while solving Nonograms, people rarely combine information from several rows and columns simultaneously, which motivates studying the simple class. A major advantage of the proposed measure is that it does not depend on the *order* in which individual rows and columns are considered. The only degree of freedom in this strategy is whether one starts with the rows or columns. This choice can make a difference of at most 1 in the resulting difficulty. An interesting property of Nonograms is that small local changes in the image can have a profound impact on the solution process for the corresponding Nonogram. The difficulty can vary wildly, by changing just a single pixel.



(a) Size 4×4 (b) Size 5×5

(c) Size 6×6 (d) Idem as (c), logarithmic

Figure 3: Number of Nonograms of a given size as a function of difficulty level. In (d), the vertical axis has logarithmic scaling.

The proposed difficulty measure can be computed efficiently, by using the *Settle* algorithm from Section 2. This allows for enumeration of a large set of Nonograms, to perform a statistical analysis of the difficulty distribution. Figure 3(a-c) shows the difficulty histogram for all simple square Nonograms of size 4×4 up to 6×6, obtained by a complete enumeration.

Note that all these Nonograms have a unique solution. It can be observed that a large fraction of all simple $n \times n$ Nonograms ($4 \leq n \leq 6$) has low difficulty (close to n), while high difficulty Nonograms (difficulty close to $\frac{1}{2}n^2$) occur rarely. For $n = 6$, out of $2^{36} \approx 7 \cdot 10^{10}$ images, 70.76 % yields a Nonogram of the simple type. (In Section 6 we will discuss the remaining ones.) Figure 3(d) shows the same histogram using a logarithmic scale. It can be observed that the average difficulty is 4.51, whereas the difficulty can be as large as 26.

Similar trends can be observed for larger Nonograms, but an exhaustive search is no longer easily possible in that case. An interesting question is how the maximum possible difficulty varies with Nonogram size. A Nonogram of high difficulty should satisfy two properties:

- In each consecutive *h-sweep* and *v-sweep* only a few new pixels should be determined. Ideally, this number of newly discovered pixel values should be bounded by a constant.
- In each consecutive *h-sweep* and *v-sweep* the value of at least one new pixel should be determined, as otherwise the Nonogram is not of the *simple* type.

For a Nonogram of size $n \times n$, $n^2 + 1$ is obviously an upper bound on its difficulty. However, it is not clear at all that the maximum difficulty that can be reached increases linearly with the number of pixels. In the next section, we will show how to construct arbitrarily large Nonograms for which the asymptotic difficulty is $\frac{1}{2}n^2$, which demonstrates that the upper bound can be attained up to a constant factor.

Before we examine these Nonograms of high difficulty, we will consider two boundary cases; proofs can be found in [3]. We first note that it is easy to see whether a line can be fully fixed by a single application of the *Settle* operation. Indeed, let $d = a_1 a_2 \dots a_k$ be a Nonogram description with $\sum_{i=1}^{k} a_i + k - 1 \leq \ell$ (which means that $?^\ell$ is fixable with respect to d). Then we have $\mathit{Settle}\,(?^\ell, d) \in \Sigma^\ell$ if and only if $\sum_{i=1}^{k} a_i + k - 1 = \ell$. This property can be easily used to characterize Nonograms of difficulty 1: all rows should have it. Finally, we characterize those situations where the *Settle* operation cannot infer any information:

Lemma *Let $d = a_1 a_2 \dots a_k$ again be a Nonogram description with $\sum_{i=1}^{k} a_i + k - 1 \leq \ell$. Then we have $\mathit{Settle}\,(?^\ell, d) = ?^\ell$ if and only if $\sum_{i=1}^{k} a_i + k - 1 \leq \ell - \max_{1 \leq i \leq k} a_i$.* □

4 Difficult Nonograms

In this section we will construct certain $m \times n$ Nonograms of the simple type that require approximately $\frac{1}{2}mn$ sweeps, thereby attaining a very high difficulty. More precisely, we show:

Theorem *Let m satisfy $m = 8k + 2$ for some integer $k \geq 1$, and take an even integer n with $n \geq 14$. Then there exists an $m \times n$ Nonogram that requires $A(m, n) = (m+2)(2n-15)/4+10$ sweeps (if $k > 1$). If $k = 1$, so $m = 10$, the Nonogram requires $6n-37$ sweeps. For square $n \times n$ Nonograms of this special type (so $n = 8k + 2$ with integer $k \geq 2$) we need $(n^2 - \frac{11}{2}n + 5)/2$ sweeps.*

The remaining part of this section is devoted to the construction of these special $m \times n$ Nonograms, and to the proof that the number of sweeps is equal to $A(m, n)$, as mentioned in the theorem. Slightly abusing notation, we will employ regular expressions for Nonogram descriptions, e.g., we will use $2(13)^2$ instead of the "official" 2 1 3 1 3; and we will write "description" instead of the more formal "Nonogram description".

Figure 4 shows the construction for $m = n = 18$. It is possible to give similar constructions for slightly varied values of m and n, for instance for odd width, but we will not go into detail on this. The slightly different value (2 less than the general formula predicts) if $k = 1$ is explained by a small case difference in the construction, see below.

The construction proceeds as follows. There are k rows with description n, i.e., consisting of only 1s. These rows, the so-called *split rows*, being the $(8i - 1)$th rows ($1 \leq i \leq k$), are fully fixed in the first *h-sweep*. Furthermore, all columns, except for the first, second and last one, have description 1^{3k+1} (where σ^r denotes a sequence of r copies of a sequence σ).

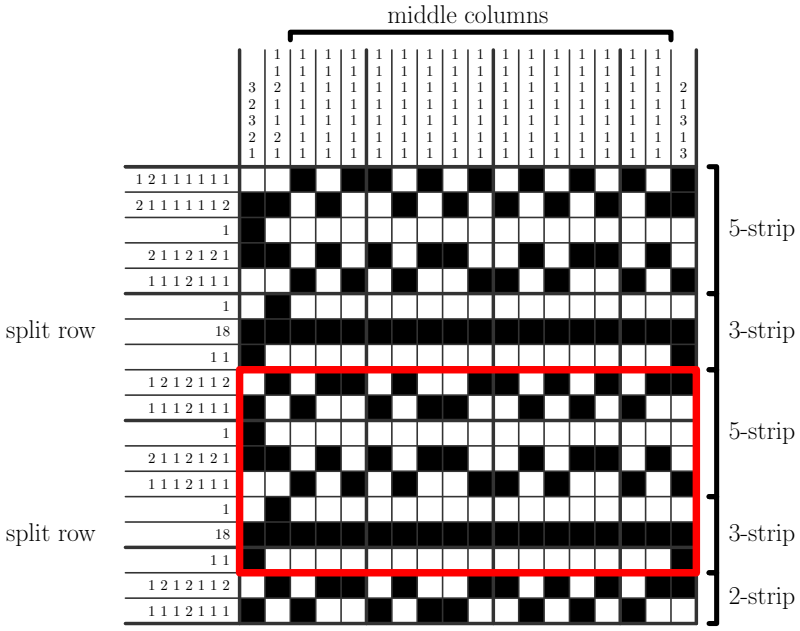


Figure 4: Overview of the construction of an 18×18 Nonogram with difficulty 115. The construction can be extended in the vertical direction by inserting consecutive copies of the marked block. Extension in the horizontal direction is straightforward.

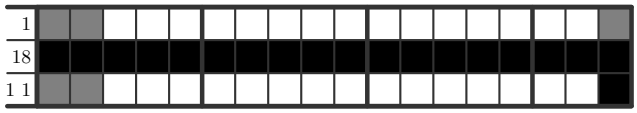


Figure 5: Contents of a 3-strip after the third sweep ($n = 18$). Gray squares denote unknown pixels.

After the first *v-sweep*, the rows immediately above and below the split rows are therefore filled with 0s in all these columns, referred to as the *middle columns*. Any three such rows together, i.e., split row and rows immediately above and below it, form a so-called *3-strip*. Each row above a split row has description 1, each row below a split row has description 1^2. The second 1 can in the third sweep also be fixed easily at the end of the row. So together any 3-strip will —after the third sweep— look like Figure 5.

Now the Nonogram is in fact separated by the 3-strips into k parts of height 5, called the *5-strips*, and a final part consisting of the bottom two rows, called the *2-strip*. All these parts must be solved in turn, as will be clear from the sequel.

Note that the 5-strips are all alike, except for the first one, which is used for bootstrapping the solver procedure. Within each 5-strip, the middle row will be filled with 10^{n-1} after the third sweep (its description is a single 1), and then the top two rows of the 5-strip will be solved, largely pixel by pixel, from right to left; after that the bottom two rows of the 5-strip will be solved in a similar fashion, from left to right, again largely pixel by pixel. The traversals from the top two rows to the bottom two rows within each 5-strip, and from each 5-strip to the next 5-strip or the final 2-strip, require special care. These traversals, combined with 5-strip solving, all invert the direction in which pixels are fixed, thereby constituting a zig-zag pattern.

The descriptions for the first, second and last columns are $(32)^k1$, $(112)^k1$ and $2(13)^k$, respectively.

Let us, to begin with, concentrate on the first (and special) 5-strip. The descriptions of its rows are $121^{n/2-3}$, $21^{n/2-3}2$, 1 (as said above), $21^{n/2-7}(21)^2$ and $1^{n/2-6}21^3$, respectively. The other 5-strips have a slightly different description for the first two rows, namely $(12)^21^{n/2-7}2$

and $1^3 21^{n/2-6}$. The row descriptions for the final 2-strip are the same: these two rows can be viewed as the top part of a regular 5-strip. In Figure 4 the resulting solved 18×18 Nonogram is shown; note the two split rows.

One can verify that after the first three sweeps, the following pixels are fixed: most pixels from the 3-strips (as mentioned above, cf. Figure 5; the five remaining unknown pixels are used for the traversals within and between the 5-strips), the bottom right pixel of the Nonogram (at 0), the two topmost pixels of the second columns from the left and right (at 01), and the entire middle row from each 5-strip (at 10^{n-1}, as said above). This last filling has the important property that in the middle columns, all 1s are now almost pinned: they must be in either first or second row, fourth or fifth row, and so on. This enforces that all 5-strips must be solved in order, and really after one another.

Concentrating on the first two rows, one can see that after the fourth sweep, only six pixels are fixed. The order, or rather the number of the sweep in which the pixels are found (again for $n = 18$; circles denote black pixels) is shown in Figure 6. Here, for each two unknown pixels immediately above one another (except for the leftmost two, where this fact is not known yet), exactly one must be 1. This is inferred pixel by pixel, coming from the right, and alternating between top and bottom row, thus contributing to the large number of sweeps needed. The 2s in the descriptions are necessary for the construction of the traversal; this also holds, in several variations, for other rows in 5-strips. Note that in the third sweep no new pixel values are found for these rows.

31	2	30	27	27	26	23	22	19	18	15	14	11	10	7	6	2	4
29	1	29	28	28	25	24	21	20	17	16	13	12	9	8	5	1	4

Figure 6: Order in which pixel values are found for the first two rows.

Finally, let us examine the number of sweeps. From the construction it is clear that the addition of *two* new columns (among the middle columns) increases the number of sweeps by $m + 2$. Indeed, in every 5-strip we need an extra 8 sweeps, and the final 2-strip adds another 4; together we get $8k + 4 = m + 2$ of them. Therefore, $A(m, n)$ should satisfy $A(m, n + 2) = A(m, n) + m + 2$.

Furthermore, it is easy to see that every extra 5-strip and its accompanying 3-strip (as shown in Figure 4) adds $4n + c$ sweeps, for some integer constant c. Careful inspection shows that $c = -30$. We conclude that $A(m, n)$ should satisfy $A(m + 8, n) = A(m, n) + 4n - 30$. Using $A(18, 18) = 115$ we arrive at the closed formula.

Note that the traversal within the first 5-strip, that reverses the right-left direction into a left-right direction, slightly differs from those in the other 5-strips. This causes the small difference in the number of sweeps for $k = 1$. □

5 Generating simple Nonograms

In this section we describe an algorithm that produces a series of Nonograms of the simple type of varying difficulty. The algorithm is rather flexible and offers many options that can be customized. We only sketch these options here. The website [8] offers an implementation.

The generated Nonograms should resemble a given gray value image $P \in \{0, \ldots, 255\}^{m \times n}$ We want these Nonograms not to look alike, and therefore maintain a set $\mathcal{L}$ of Nonograms from which a newly generated Nonogram should differ. The newly found Nonogram is then appended to $\mathcal{L}$.

As a subroutine, the algorithm for generating Nonograms uses a straightforward generalization of the *Difficulty* algorithm from Figure 2, referred to as *FullSettle*: instead of the difficulty, the *FullSettle* operation returns the set of unknown pixels, where we let the sweeps continue until they make no further progress. (This is equivalent to saying that there is no single *Settle* operation that reveals an unknown pixel.) Note that in Section 2 the *Difficulty*

algorithm was applied to Nonograms of the simple type, where the algorithm terminates by definition, whereas in the current application the Nonograms may not be solved, and termination of *FullSettle* is effected when a sweep does not yield any new fixed pixels.

Furthermore, a function *Init* (P) is used, that returns a 0–1 Nonogram that somehow resembles P, e.g., by applying a threshold operation or a binary edge detection filter to the gray level input image.

Generate $(P, \mathcal{L})$:
  $p \leftarrow$ *Init* (P); $U \leftarrow$ *FullSettle* (p);
  while $U \neq \varnothing$ **do** $p \leftarrow$ *Adapt* $(p, U, P, \mathcal{L})$; $U \leftarrow$ *FullSettle* (p); **od**
  return $(p,$ *Difficulty* $(p))$;

Figure 7: Algorithm that generates a uniquely solvable Nonogram and its difficulty.

Pseudo-code for the algorithm *Generate* is shown in Figure 7. The main ingredient is the function *Adapt* $(p, U, P, \mathcal{L})$ that returns a Nonogram p' that is equal to p, except for (at least) one pixel that is 0 in p but is 1 in p'. Note that, since the number of black pixels strictly increases, the loop in *Generate* indeed terminates: an all black Nonogram is certainly uniquely solvable. Also note that upon entering *Adapt* at least one $(i, j) \in U$ satisfies $p_{ij} \neq 0$; indeed, if *FullSettle* $(p) \neq \varnothing$, it cannot be the case that all the unknown pixels must be 1. The function *Adapt* proceeds as in Figure 8.

Adapt $(p, U, P, \mathcal{L})$:
  $min \leftarrow \infty$;
  for all $(i, j) \in U$ (in random order) **do**
    if $p_{ij} = 0$ **then**
      $p_{ij} \leftarrow 1$; `% try new image, that differs in one pixel`
      $value \leftarrow \alpha \cdot |$*FullSettle* $(p)| + \beta \cdot P_{ij} + \gamma \cdot \sum_{L \in \mathcal{L}} L_{ij}$;
      if $value < min$ **then** $min \leftarrow value$; $(k, \ell) \leftarrow (i, j)$; **fi**
      $p_{ij} \leftarrow 0$; `% restore original image`
    fi
  od
  $p_{k\ell} \leftarrow 1$;
  return p;

Figure 8: Algorithm that slightly adapts an image p.

Here suitable non-negative parameters α, β and γ must be chosen. So we want the Nonogram to have a small amount of unknowns, we would like the changed pixel to be dark in the original image, and many Nonograms from $\mathcal{L}$ to be white in that particular pixel. If $\alpha = \gamma = 0$, the final Nonogram will resemble the original P, but will usually be quite dark. However, if $\beta = 0$, resemblance will be worse. High γ-values ensure diversity. Clearly, in particular if $\mathcal{L}$ is large, it might be hard or even impossible to guarantee that the generated Nonogram sufficiently differs from those in $\mathcal{L}$.

In this way we get Nonograms of different difficulty, but usually quite hard ones. In order to obtain Nonograms of more varying and usually lower difficulty, the algorithm from Figure 9 can be used. It returns a set of at most *depth* uniquely solvable Nonograms together with their difficulties, whose sets of black pixels strictly include that of the original p, and can therefore in general be expected to have lower difficulty. Note that each Nonogram added to $\mathcal{M}$ has at least one black pixel more than its predecessor. In fact, in practice uniquely solvable Nonograms are encountered in nearly every iteration.

Figure 10 contains some examples. All pictures are of size 30×38. The first picture is the original gray value image, from which the second is obtained by thresholding (aiming at 35 % black pixels). For the third picture, an edge detection filter is applied. For the fourth picture, empty lines were addressed (in Figure 10 this visible near the ear). The pictures in the middle row are Nonograms of the simple type that have been generated consecutively by the Generate algorithm starting from the third picture in the top row, and can therefore

Vary $(p, P, \mathcal{L}, \mathit{depth})$:
  $\mathcal{M} \leftarrow \varnothing$; $d \leftarrow 0$;
  while $d < \mathit{depth}$ **and** p has white pixels **do**
    $\mathit{min} \leftarrow \infty$; $U \leftarrow \{$white pixels in $p\}$;
    for all $(i, j) \in U$ (in random order) **do**
      $p_{ij} \leftarrow 1$; $\mathit{value} \leftarrow \alpha \cdot |\mathit{FullSettle}\,(p)| + \beta \cdot P_{ij} + \gamma \cdot \sum_{L \in \mathcal{L}} L_{ij}$;
      if $\mathit{value} < \mathit{min}$ **then** $\mathit{min} \leftarrow \mathit{value}$; $(k, \ell) \leftarrow (i, j)$; **fi**
      $p_{ij} \leftarrow 0$;
    od
    $p_{k\ell} \leftarrow 1$; $d \leftarrow d + 1$;
    if $|\mathit{FullSettle}\,(p)| = 0$ **then** $\mathcal{M} \leftarrow \mathcal{M} \cup \{(p, \mathit{Difficulty}\,(p))\}$; **fi**
  od
  return $\mathcal{M}$;

Figure 9: Algorithm that generates Nonograms of varying difficulty.

expected not to look alike entirely; the numbers indicate the difficulties; the parameters of *Generate* were set at $\alpha = \gamma = 8$ and $\beta = 1$. The pictures in the bottom row are obtained from those immediately above them by the *Vary* algorithm with $\mathit{depth} = 60$; the final Nonogram generated in the main loop is depicted (other ones could also have been chosen). The set $\mathcal{L}$ contains the Nonograms from the second line created so far. Note that usually the difficulty decreases steadily during this process, but certainly not always.

Figure 10: Nonograms of Alan Turing (1912–1954). The theme "Enigma" seems appropriate.

6 . . . and beyond

In this section we will mention some issues related with non-simple Nonograms. The presentation is based on [3], but here we will restrict ourselves to examples and graphs to illustrate our points.

Let us first address other solving strategies. In the example from Figure 11a, *Settle* operations do not provide any further information (note the row with the empty Nonogram description ϵ, which is only adhered to by a string with five 0s). Therefore, the Nonogram is not of the simple type. It is, however, uniquely solvable. This can even be proved by straightforward logic. For the four pixels a, b, c and d in the bottom left corner one can deduce a dependency graph as in Figure 11b, where, e.g., an arrow from d to $\neg b$ means that if d is black, b must be white. This follows from the description of the second column. Equivalently, one can look at the corresponding Boolean formula, part of it being $(\neg d \vee \neg b)$. Indeed, we arrive at a 2-SAT problem in this way.

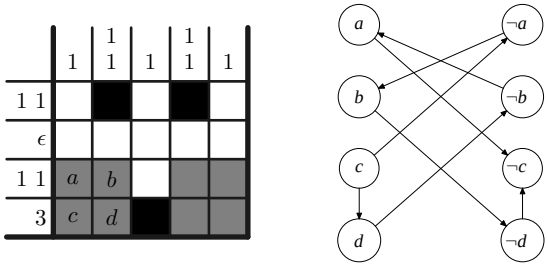


Figure 11: (a) Partially solved Nonogram; (b) (part of) its corresponding dependency graph.

We can now 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. For the example from Figure 11a we can infer that c is `false` (or 0); now a few more *Settle* operations suffice to solve the puzzle.

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. Note that it is indeed possible to apply general SAT- or CSP-solvers to the problem of solving Nonograms; see [3] for some more information.

In Section 3 all Nonograms of small sizes were enumerated, see Figure 3. Figure 11 presented a small Nonogram of non-simple type, where its solution required some more sophisticated arguments. However, as the example from Figure 12 shows, small Nonograms can still be harder. The above-mentioned 2-SAT approach does not yield any progress here.

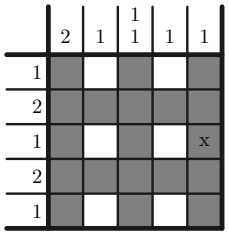


Figure 12: Partially solved 5×5 Nonogram, where the fact that pixel `x` must be white is hard to infer. The gray pixels are still unknown.

It is clear that different levels of solvers exist, the one that by definition solves simple Nonograms (called *FullSettle* in Section 5) being the easiest. Figure 13 and Figure 14 provide some statistical information regarding this issue, where different percentages of black pixels and puzzle sizes are addressed. Here, *Solver0* applies a combination of the *Settle* operation and the 2-SAT approach, as described above. And *Solver1* furthermore also allows for single "guesses" of the value of a pixel, which in case of a contradiction can be fixed to the other value.

We finally mention some issues concerning (non-)solvability of Nonograms, and we pay special attention to puzzles with multiple solutions. Nonograms (not those in puzzle collections, of course) can have more than one solution. Figure 15a shows a small example, referred to as an *elementary switching component*: four pixels, two black and two white, such that interchanging the black and the white pixels does not change the description. However, where the non-uniqueness problem for Discrete Tomography allows an elegant description based

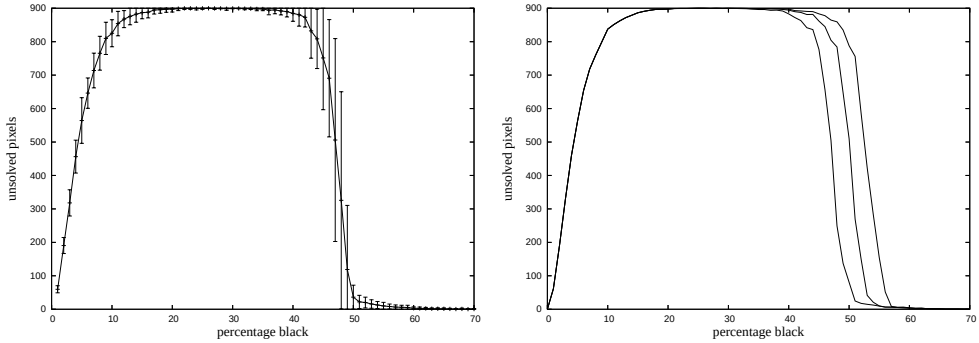


Figure 13: (a) Average number of unsolved pixels for randomly generated 30×30 puzzles, for a varying percentage of black pixels, when using *Solver1*; error bars indicate the standard deviation (100 runs for every percentage). (b) As (a), for *FullSettle* (top graph), *Solver0* (middle graph) and *Solver1* (bottom graph), without standard deviation.

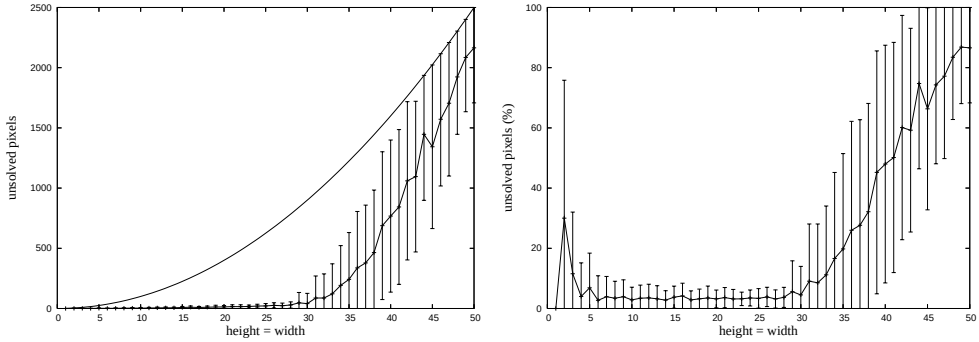
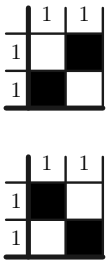
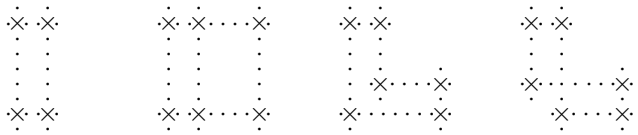


Figure 14: (a) Average number of unsolved pixels for randomly generated puzzles of different size, with a fixed percentage of 50 % black pixels, again using *Solver1*; error bars indicate the standard deviation. The smooth curve is the total number of pixels. (b) As (a), but now showing these values as percentage of the total number of pixels.



(a) Elementary switching component with its two solutions



(b) More complex types of switching components for Nonograms

Figure 15: Switching components in Nonograms, containing up to six unknowns.

on these elementary switching components [11], the non-uniqueness problem for Nonograms appears to be much more complex. The problem of deciding whether *another* solution of a Nonogram exists, given a particular solution, is NP-hard [17].

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 (Figure 15b, left), similar to an elementary switching component. 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. And we have precisely 2 solutions here. Between the two unknowns we must have only 1s, or a series of solitary 1s with variable length blocks of 0s in between: in regular expression notation $(0^+1)^*0^+$.

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 (Figure 15b, right). In the former case we have 3 solutions, where in between two unknowns we can only have the $(0^+1)^*0^+$ situation, in the latter case there are precisely 2 solutions — as in the $u = 4$ case. In all these situations, unknowns cannot touch.

7 Conclusions and further research

Nonograms are interesting study objects, due to their links with both combinatorial optimization and logic reasoning, as well as their rich variety of combinatorial properties. In this paper, we focused on the set of *simple* Nonograms, which can be solved by a series of reasoning steps involving only a single column or row at a time. We proposed a *difficulty measure* for this class, which corresponds roughly with the solution strategy followed by human puzzlers and has favourable computational properties.

After studying the basic notions, we described a family of Nonograms that have asymptotically maximal difficulty, up to a constant factor. An interesting question remains if the difficulty can still be increased to cn^2, where $c \in (\frac{1}{2}, 1]$. Next, we briefly described an algorithm for *generating* Nonograms of varying difficulty. The basic steps of this algorithm allow for a broad spectrum of variants, each yielding different types of Nonograms. Finally we mentioned some issues related to non-simple Nonograms. We intend to explore these in future work.

References

[1] K.J. Batenburg, S. Henstra, W.A. Kosters, and W.J. Palenstijn. Constructing simple Nonograms of varying difficulty. *Pure Mathematics and Applications (Pu.M.A.)*, 20:1–15, 2009.

[2] K.J. Batenburg and W.A. Kosters. A discrete tomography approach to Japanese puzzles. In *Proceedings of the 16th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC)*, pages 243–250, 2004.

[3] K.J. Batenburg and W.A. Kosters. Solving Nonograms by combining relaxations. *Pattern Recognition*, 42:1672–1683, 2009.

[4] T.M. Buzug. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer, 2008.

[5] G.T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Springer, second edition, 2010.

[6] G.T. Herman and A. Kuba. *Discrete Tomography: Foundations, Algorithms and Applications*. Birkhäuser, 1999.

[7] G.T. Herman and A. Kuba. *Advances in Discrete Tomography and its Applications*. Birkhäuser, 2007.

[8] W.A. Kosters. Website Nonogram [accessed 9.2.2012], `www.liacs.nl/~kosters/nono/`, 2012.

[9] E.G. Ortiz-García, S. Salcedo-Sanz, J.M. Leiva-Murillo, Á.M. Pérez-Bellido, and J.A. Portilla-Figueras. Automated generation and visualization of picture-logic puzzles. *Computers and Graphics*, 31:750–760, 2007.

[10] E.G. Ortiz-García, S. Salcedo-Sanz, Á.M. Pérez-Bellido, L. Carro-Calvo, A. Portilla-Figueras, and X. Yao. Improving the performance of evolutionary algorithms in grid-based puzzles resolution. *Evolutionary Intelligence*, pages 169–181, 2009.

[11] H. J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.

[12] S. Salcedo-Sanz, E.G. Ortiz-García, Á.M. Pérez-Bellido, J.A. Portilla-Figueras, and X. Yao. Solving Japanese puzzles with heuristics. In *Proceedings IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 224–231, 2007.

[13] S. Salcedo-Sanz, J.A. Portilla-Figueras, E.G. Ortiz-García, Á.M. Pérez-Bellido, and X. Yao. Teaching advanced features of evolutionary algorithms using Japanese puzzles. *IEEE Transactions on Education*, 50:151–156, 2007.

[14] S. Simpson. Website Nonogram solver [accessed 9.2.2012], `www.comp.lancs.ac.uk/~ss/nonogram/links.html`, 2011.

[15] J.-T. Tsai. Solving Japanese nonograms by Taguchi-based genetic algorithm. *Applied Intelligence*, 2012 [to appear].

[16] J.-T. Tsai, P.-Y. Chou, and J.-C. Fang. Learning intelligent genetic algorithms using Japanese nonograms. *IEEE Transactions on Education*, 2012 [to appear].

[17] N. Ueda and T. Nagao. NP-completeness results for Nonogram via parsimonious reductions, preprint, 1996.

[18] J.K. Vis, W.A. Kosters, and K.J. Batenburg. Discrete tomography: A neural network approach. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC)*, pages 328–335, 2011.

[19] Website Griddlers [accessed 24.2.2012], `www.griddlers.net`, 2012.

[20] G.J. Woeginger. The reconstruction of polyominoes from their orthogonal projections. *Information Processing Letters*, 77:225–229, 2001.