Universiteit Leiden
The Netherlands

Opleiding Informatica

Analysis of nonsimple Nonograms

Philippe P. Bors

Supervisors:
dr. Walter A. Kosters
dr. Hendrik J. Hoogeboom

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

??/07/2019

Abstract

Nonograms are a type of logic puzzles in which the puzzler is supposed to fill in the correct values for all cells in a rectangle grid, effectively composing a pixelated image. The lengths of the consecutive segments of non-empty pixels in each row and column should, preserving the order in which these segments appear, match their corresponding puzzle line descriptions.

In general, puzzles can have an arbitrary number of pixel values (colours). In this thesis we focus on Nonograms with only two possible pixel values, depicting a black and white binary image. Firstly, several solving strategies are explained and examined. Then, the characteristics of these puzzles and strategies are analyzed by mass solving large sets of puzzles with relatively small image spaces, as well as by taking samples for larger puzzle sizes. Finally, we propose a difficulty measure to classify the puzzles and try to capture some of the characteristics that can make a puzzle substantially harder to solve.

Contents

1 Introduction

The first logical puzzles date back as far as the 19th century, in which mathematician Lewis Carroll used logic reasoning on premises to create a new type of game [Car86]. Although popular in its era, most logic puzzles known by the modern-day public are not nearly as verbose as their predecessors. This is all due to the upcoming of the logic grid puzzle in the last few decades, such as the logic maze and Sudoku.

Originating from Japan and known under many different names such as Paint by Numbers, Griddler and Japanese puzzles, Nonograms have especially proven popular in East-Asia where they still frequently appear in newspapers. Puzzles of higher difficulty are usually distributed in puzzle magazines around the globe.

Previous research on Nonograms spans many different fields of computing. Chiung-Hsueh et al. [YLC11] use chronological backtracking to cope with known deficiencies in precursory proposed algorithms. Salcedo-Sanz et al. [SSOGPB+07] compare solving strategies based on constraint programming with a genetic algorithm. Batenburg and Kosters [BK12] propose a (p, q)-Solver which takes into account p rows and q columns per step to find logical contradictions in the resulting $p \cdot q$ intersections. Apart from analyzing solving strategies, some work has been done in constructing Nonograms as well. Batenburg et al. [BHKP09] show the effectiveness of an algorithm that generates a simple Nonogram of varying difficulty based on a given gray level image. Ortiz-García et al. [OGSSLM+07] do the same to some extent by proposing an algorithm capable of constructing a series of logic grid puzzles from any RGB color image.

In this thesis we first explain the basic concept of a Nonogram in Chapter 2. Then, we propose three primary solving strategies and explain their algorithmic procedures in Chapter 3, as well as by giving some formulated examples. These strategies have been implemented in two programs written in C++14; one optimized for solving individual puzzles, the other for solving and analyzing large sets of Nonograms given a specific size. The latter program is actually an extension and partial reimplementation of a program created and provided to me by one of my supervisors. This thesis also defines a difficulty measure for Nonograms and categorizes all puzzles up to size 6×6 by difficulty. Later, in Chapter 3.2.5, we analyze a special case of the (p, q)-Solver where $p = q = 2$, introduced by Batenburg et al. as the FourSolver, and compare it with our own solving strategy based on 2-satisfiability.

This thesis is the result of a bachelor project under the supervision of W.A. Kosters and H.J. Hoogeboom from the Leiden Institute of Advanced Computer Science (LIACS), the computer science department of Leiden University.

2 Definitions

Let m denote the height and n denote the width of the puzzle. First we define pixel alphabet $\Sigma = \{\square, \blacksquare\}$ (also referred to as white and black pixels) and extended pixel alphabet $\Gamma = \Sigma \cup \{?\}$, where '?' denotes the unknown pixel value. A *line description* is an ordered sequence $(d_1, d_2, \ldots, d_\ell)$ with $d_i \in \mathbb{N}^+$ $(i \in \{1, 2, \ldots, \ell\})$ and $0 \leq \ell \leq k-1$ where $k = m$ for vertical line descriptions and $k = n$ for horizontal line descriptions. Moreover, for every description it should hold that $\sum_{i=1}^{\ell} d_i + \ell - 1 \leq k$. Let $\mathcal{D} = \langle \mathcal{D}_H, \mathcal{D}_V \rangle$ be a 2-tuple containing the sets of horizontal and vertical line descriptions, respectively. Hence $|\mathcal{D}_H| = m$ and $|\mathcal{D}_V| = n$ in every case. We denote the i^{th} horizontal and vertical line descriptions by $\mathcal{D}_H(i)$ and $\mathcal{D}_V(i)$ respectively.

A Nonogram *puzzle grid* $\mathcal{P}$ is a set-up of ordered sequences (rows or columns) of equal cardinality $s_1, s_2, \ldots, s_k \in \{(a_1, a_2, \ldots, a_q) \mid a_i \in \Gamma \ (i \in \{1, 2, \ldots, q\}) \wedge q = k'\}$, together representing a field of $m \times n$ pixel values like a two-dimensional array. Here, k' denotes the counterpart value of k, defined as $k' = m + n - k$. For convenience, let $\mathcal{P}_H$ $(k = m)$ and $\mathcal{P}_V$ $(k = n)$ be the same grid represented by horizontal and vertical lines, respectively. If defined analogously to $\mathcal{D}$, we indicate the i^{th} row and column by $\mathcal{P}_H(i)$ and $\mathcal{P}_V(i)$. A *puzzle image* or simply image is a grid $\mathcal{P}$ such that for all pixels it holds that $\mathcal{P}_H(i)(j), \mathcal{P}_V(j)(i) \in \Sigma$ $(1 \leq i \leq m, 1 \leq j \leq n)$. Put differently, this is a grid where all the ?'s are replaced by □'s and ■'s. If $\mathcal{P}$ is an image, it is a potential *solution* to the puzzle. Furthermore, $\mathcal{P}$ is called a solution to the puzzle if all the rows and columns adhere to their respective descriptions. A sequence (row or column) s *adheres* to a description if the lengths, in order, of the ■-subsequences in s match the ones given in the description. These subsequences are separated by at least one □.

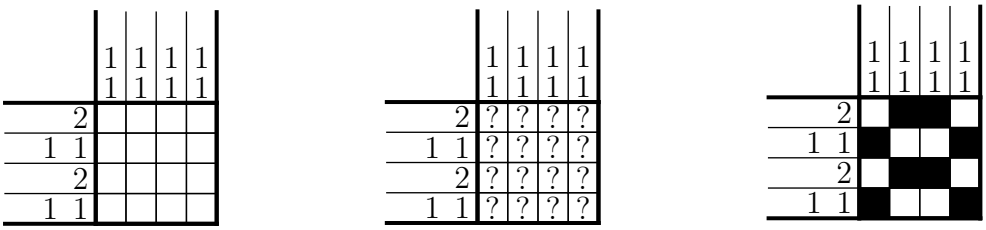


Figure 1: Empty 4×4 Nonogram as presented to the puzzler as challenge (a); the same puzzle in the 0-configuration, clearly differentiating white pixel values from unknown ones (b); solution to the puzzle (c).

Finally, a Nonogram $\mathcal{N}$ is a 2-tuple $\mathcal{N} = \langle \mathcal{P}, \mathcal{D} \rangle$ where $\mathcal{P}$ is the puzzle grid and $\mathcal{D}$ is are the puzzles' line descriptions. Simply put, a Nonogram is a logic puzzle, consisting of a $m \times n$ pixel grid and $m + n$ line descriptions. Each row and column has exactly one line description. The puzzler starts off with a (possibly) empty grid, to be filled with pixels adhering to the line descriptions for the rows and columns simultaneously. Although puzzles usually begin with an empty grid, it is not uncommon for puzzles of higher difficulty to start with *clues*; these are mostly given black pixel values. In Figure 1, one could argue that there exists some level of ambiguity between an ordinary unknown (yet to be determined) pixel and an actual white pixel (□) in (a). To remove this ambiguity we introduce puzzles in their 0-configuration. In this configuration, the pixels of which the values are unknown are marked by their actual value '?', rather than leaving aside if they are simply unknown or white.

2.1 Unique solutions

A puzzle is called *uniquely solvable* iff there is exactly one image that adheres to its line descriptions. This means that there exists only one image in the entire image space of $m \times n$ puzzles, holding a total of $2^{m \cdot n}$ images, that fits the descriptions of the puzzle. The puzzle in Figure 2 is the simplest example of a puzzle that is not uniquely solvable. Naturally, if $\mathcal{N}$ is uniquely solvable, then the puzzles corresponding to its possible rotations (maximum of 8) are so as well. Uniquely solvable puzzles are usually the easiest class of Nonograms. This is due to their characteristics, and shall be explained in the next chapter when we introduce the *nonsimple* concept.

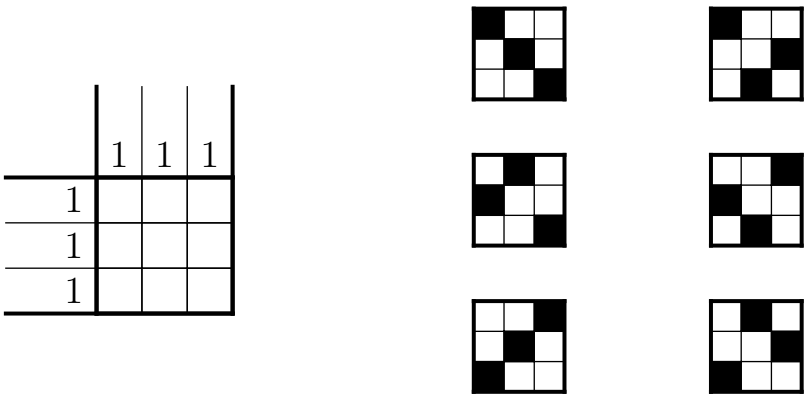


Figure 2: Nonuniquely solvable 3×3 Nonogram with all 6 possible solutions.

3 Solving strategies

In this chapter we introduce three solving strategies and explain their algorithmic procedures. These algorithms form the basis for the difficulty classification in the next chapter.

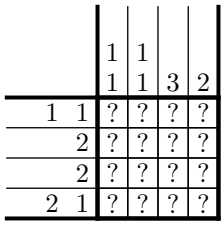
3.1 Simple solving

When looking at individual puzzle lines and their descriptions only, one can derive pixel values for which there is no doubt they hold a certain value. Let $\mathcal{L} = (a_1, a_2, \ldots, a_k)$ be a puzzle line with its description $\mathcal{D}$. If, for every possible placement of pixel values to which $\mathcal{L}$ adheres to $\mathcal{D}$, any $a_i \in \mathcal{L}$ where $\mathcal{L} \subseteq \Gamma$ and $a_i \neq ?$ has the same value for *every* possible line fix, we have found the definitive value of a_i. Loosely speaking, for a single line we fill in all pixel values that can be derived from individual line descriptions only, by the process of elimination. This is what we call the SimpleSolver algorithm (see Algorithm 1). It works by performing a series of horizontal and vertical *sweeps*: carrying out the procedure described as above for every row (H-Sweep) or column (V-Sweep), respectively. Accordingly, when SimpleSolver has carried out its procedure on a puzzle, all pixel values that could be derived by a series of very simple logical reasoning steps are placed in the puzzle grid.

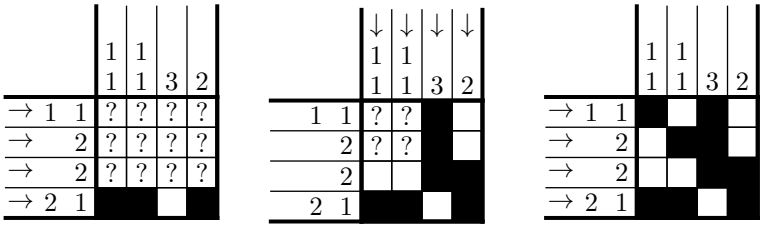
Algorithm 1: SimpleSolver

```
   begin
1      while ¬puzzle.solved do
2          H-Sweep(puzzle)
3          if ¬puzzle.solved then
4              V-Sweep(puzzle)
```

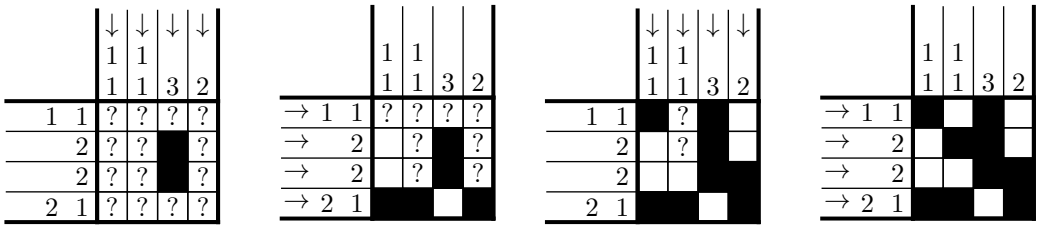
This is a very basic outline of the procedure performed by the SimpleSolver algorithm. Note that the loop starts with an H-Sweep in line 2, rather than a vertical sweep. Changing the order in which the sweeps occur (toggling the starting sweep) can make a difference in terms of the amount of sweeps required to solve the puzzle. But this difference will never be larger than 1, because even though H-Sweeps and V-Sweeps are independent of one another, they can discover the same pixel values. Hence, the situations in which this is not the case can cause one extra sweep search in the beginning, but because the sweep procedures are interleaved, the missing value that was bound to be discovered will certainly be found in the next corresponding sweep.



(a) 0-configuration



(b) Progress with horizontal starting sweep



(c) Progress with vertical starting sweep

Figure 3: Differences in sweep counts can occur.

Figure 3 illustrates the progress made after every consecutive sweep in the algorithm, starting with an H-Sweep at (b). The puzzle from (a) is finally solved in the third panel. However, starting with a vertical sweep in (c) results in one extra sweep in comparison to the procedure in (b). This particular example is exactly what we call a *simple* Nonogram: a puzzle that can be solved by SimpleSolver. Nonograms of this class are the easiest to solve. No guessing is involved in the process, nor are there any clues required to solve the puzzle. The number of sweeps required to completely solve the puzzle starting from the 0-configuration is the difficulty measure for simple puzzles. In this particular example, we obtain two different sweep counts from changing the starting sweep. Therefore, we take the average of these two numbers. The difficulty of this puzzle is then denoted by $\mathcal{S}_{3.5}$, describing a simple puzzle that is solved in (an average of) 3.5 sweeps. Simple puzzles are always uniquely solvable, because if they were not, this algorithm would not be able to solve them since it can only detect pixel values that have a fixed (and therefore unique) position.

3.2 2-SAT solving

In this subchapter we introduce a solving strategy based on 2-satisfiability, as well as a polynomial algorithm that can determine if a puzzle encapturing all constraints in 2-CNF has ≥ 1 solution(s). Some basic notions about 2-satisfiablity are given in an introduction, and we further apply this to Nonograms in Chapter 3.2.2. An algorithm from Batenburg et al. [BK12] is scrutinized and compared to the newly introduced solving strategy, which in turn shall be shown to be a complete version of the fragmentary approach of its counterpart.

3.2.1 Introduction to 2-satisfiability

The Boolean satisfiability problem (SAT abbreviated) copes with determining if there exists a mapping $\beta\colon \mathcal{X} \mapsto \{\texttt{True}, \texttt{False}\}$ for the Boolean variables $\mathcal{X} = \{x_1, x_2, \dots x_n\}$ in a given formula such that β *satisfies* this formula ϕ [BFO+10]. In other words, this means that the standard Boolean evaluation of ϕ yields the value `True`, using assignment β. A formula ϕ is said to be in *conjunctive normal form* (CNF abbreviated) if the formula consists of a conjunction of disjunctions of literals, like $\bigwedge_{i=1}^{m} C_i$, where the C_is denote *clauses* (e.g. $(x_a \vee x_b)$). A literal is simply some variable $x \in \mathcal{X}$ or its negation $\overline{x}$. Notice that 2-SAT is a special case of SAT where the formula ϕ is usually given in 2-CNF as follows:

$$\phi \stackrel{\text{def}}{=} (\varphi_1 \vee \psi_1) \wedge (\varphi_2 \vee \psi_2) \wedge \ldots \wedge (\varphi_n \vee \psi_n)$$

Where $\varphi_1, \varphi_2, \dots, \varphi_n$ and $\psi_1, \psi_2, \dots, \psi_n$ denote literals over a finite set of variables $\mathcal{X}$ such that $\{\varphi_1, \psi_1\} \cup \{\varphi_2, \psi_2\} \cup \ldots \cup \{\varphi_n, \psi_n\} = \bigcup_{i=1}^{n} \{\varphi_i, \psi_i\} \subseteq \mathcal{X} \cup \{\overline{x} \mid x \in \mathcal{X}\}$.
The general satisfiability problem k-SAT is an $\mathcal{NP}$-complete problem for $k \geq 3$, but 2-SAT $\in \mathcal{P}$. The complexity class $\mathcal{P}$ consists of all decision problems that can be solved in a polynomial amount of computing time, using a deterministic Turing machine, whereas $\mathcal{NP}$-complete problems are the most difficult problems in the complexity class $\mathcal{NP}$, known to hold all decision problems for which their yes-instances can be proofed in polynomial time. Hence 2-SAT can actually be solved by algorithms that run in polynomial time, but clauses consisting of 3 or more literals cant. Apart from actual 2-SAT-solving algorithms, determining if a formula is *satisfiable* can also prove useful. A formula is said to be satisfiable if there exists ≥ 1 truth-assignment β over $\mathcal{X}$ such that the Boolean evaluation of the formula is `True`. Multiple algorithms for solving 2-SAT and determining if a formula in 2-CNF is satisfiable exist, including the Kosaraju-Sharir algorithm implemented in Chapter 3.2.3.

3.2.2 Nonograms and 2-satisfiability

When we turn our attention to Nonograms, the relation between these puzzles and 2-SAT becomes evident if we apply the notion of Boolean variables to unknown pixel values where `True` = ■ and `False` = □. Using the line descriptions and resolved pixels of the puzzle, we can construct a formula ϕ in 2-CNF that represents the topological constraint problem of the Nonogram in question. For a simple puzzle line $\mathcal{L} = (?, \square, ?, ?, \square)$ with description $(1, 1)$, we could, for example, argue that if the third unknown pixel value is set to black, the definitive value of the fourth pixel should be white, since the line description only allows consecutive black segments of size 1. Moreover, setting the third pixel to black also implies that the first pixel should hold this value, since that would be the only possible placement of the remaining black pixel that is part of two segments of consecutive black pixels, when the third pixel value is white. If we apply this procedure to all rows and columns of a Nonogram $\mathcal{N}$, we end up with a set of logical disjuntions over the variables, being the formula $\phi(\mathcal{N})$. Because SAT-solvers usually run in some polynomial time based on the number of clauses and variables in the formula, it seems natural to first apply SimpleSolver to a Nonogram to remove as much variables as possible by fixing these values in polynomial time as well. Fewer unknown pixel values mean fewer variables and therefore smaller formulæ to solve. In theory, for nonsimple Nonograms, this relieves a potential 2-SAT algorithm that would continue where SimpleSolver left off.

However, it proves more useful to look at implicative forms rather than clauses following 2-CNF restrictions. Not only is it more comprehensible to reason about pixel relations this way, it is also convenient for building *implication graphs*.

		1 1	2	1 1	1
1	1	■		x_1	x_2
	1			x_3	x_4
	2	■	■		
	2		■	■	

$$(\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3)$$
$$\wedge \ \ldots \ \wedge$$
$$(\overline{x_4} \vee \overline{x_2}) \wedge (\overline{x_4} \vee \overline{x_3}) \wedge (x_4 \vee x_2) \wedge (x_4 \vee x_3)$$
$$\equiv$$
$$(x_1 \Rightarrow \overline{x_2}) \wedge (x_1 \Rightarrow \overline{x_3}) \wedge (\overline{x_1} \Rightarrow x_2) \wedge (\overline{x_1} \Rightarrow x_3)$$
$$\wedge \ \ldots \ \wedge$$
$$(x_4 \Rightarrow \overline{x_2}) \wedge (x_4 \Rightarrow \overline{x_3}) \wedge (\overline{x_4} \Rightarrow x_2) \wedge (\overline{x_4} \Rightarrow x_3)$$

Figure 4: Nonuniquely solvable Nonogram with derivable formula in 2-CNF.

These directed graphs (digraphs) encapture the implicative relations between variables and are used by the algorithms introduced in the following two subchapters. They generally consist of $2 \cdot n$ vertices $\mathcal{V} = \bigcup_{i=1}^{n} \{v(x_i), v(\overline{x_i})\}$ with n being the number of variables (unknown pixel values in the constraint problem) and m edges $\mathcal{E}$ with m being the number of clauses of the formula ϕ in implicative normal form and v a function $v\colon \mathcal{X} \cup \{\overline{x} \mid x \in \mathcal{X}\} \mapsto \{v_1, v_2, \ldots, v_n\}$ that maps every literal $\ell \in \{\mathcal{X} \cup \{\overline{x} \mid x \in \mathcal{X}\}\}$ to some vertex v.

Consider the example in Figure 4. This partially solved puzzle leaves four pixel values unresolved, effectively creating the same constraint problem as seen in Figure 2 in Chapter 2.1, but on a smaller scale. The implicative relations between these four consecutive pixels allow multiple solutions; two in this case. It can be considered as a nonuniquely solvable puzzle that is responsible for the ambiguous solution of the entire puzzle. This is what we call a *switching component*.

It is important to note that altogether, 2-SAT does not encapture every single relation between pixel values. Because clauses with ≥ 3 variables are not uncommon in larger puzzles, i.e., $(x_1 \vee x_2 \vee x_3) \equiv (\neg(x_1 \vee x_2) \Rightarrow x_3)$, we know the resulting fomulæ of these problems will certainly not be in 2-CNF and therefore not entirely solvable by applying 2-satisfiability in polynomial time.

3.2.3 Kosaraju-Sharir

If all the implicative relations of a puzzle can be represented by a formula ϕ in 2-CNF, it is possible to determine, for this specific instance, if there exists a satisfying truth-assignment for the variables. For this, we will be using the Kosaraju-Sharir algorithm. Independently discovered by both, but published by Sharir [Sha81] in 1981, this algorithm is designed to find the *strongly connected components* in a graph. A strongly connected component (SCC abbreviated) of a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ is a subset $\mathcal{SCC} \subseteq \mathcal{V}$ such that $\forall v, u \in \mathcal{SCC} : (v \neq u)\ ([v \to u] \in \mathcal{E}^+)$ where $\mathcal{G}^+ = \langle \mathcal{V}^+, \mathcal{E}^+ \rangle$ is the transitive closure of $\mathcal{G}$; formulated otherwise, there exists a path from v to u in $\mathcal{G}^+$ for all pairs of two nodes in some SCC. This means that all vertices in a SCC are connected internally (are reachable).

Kosaraju-Sharir uses a recursive subroutine, for instance depth-first search, to find a topological ordering $\mathcal{T}(\mathcal{G})$ of the original graph $\mathcal{G}$: a sequence of all vertices such that if there exists a path $[u \to v]$, u is found before v in the ordering. The algorithm then assigns vertices to SCCs by walking through the transposed graph $\mathcal{G}^T$ in the previously

found order using the same ordering subroutine as before. To understand why this works, let $\overrightarrow{\mathcal{V}}(u)$ be all vertices reachable from u by only forward traversal, $\overleftarrow{\mathcal{V}}(u)$ defined analogously for backward traversal, and $\mathcal{Q}(u) = \{a_i \mid a_j = u \wedge i < j\} \subsetneq \mathcal{V}$ for any topological order $\mathcal{T}(\mathcal{G}) = (a_1, a_2, \ldots, a_n)$ would be all vertices appearing before u in $\mathcal{T}(\mathcal{G})$. Naturally, $\overleftarrow{\mathcal{V}}(u) \cap \overrightarrow{\mathcal{V}}(u) = \overleftarrow{\mathcal{V}}(u) \setminus (\overleftarrow{\mathcal{V}}(u) \setminus \overrightarrow{\mathcal{V}}(u)) = \overleftarrow{\mathcal{V}}(u) \setminus \mathcal{Q}(u)$ would be the strongly connected component considering u to be the root node. Even though slightly more efficient variations of this algorithm exist, we will be using a version with depth-first search as its recursive subroutine to order the vertices because of its relative simple approach compared to breadth-first search and other techniques. In general, Kosaraju-Sharir makes the two traversals through the graphs in linear $\Theta(|\mathcal{V}| + |\mathcal{E}|)$ time.

Our algorithm will make use of the fact that if $v(x)$ and $v(\overline{x})$ are in the same strongly connected component, both vertices can visit one another, making a solution impossible and ϕ unsatisfiable because $(x \Rightarrow \overline{x}) \wedge (\overline{x} \Rightarrow x) \equiv (\overline{x} \vee \overline{x}) \wedge (x \vee x) \equiv (\overline{x} \wedge x)$ has no solution. Algorithm 2 is an implementation of the algorithm using Kosaraju-Sharir to determine if an instance of 2-SAT has a satisfiable solution.

Note that this algorithm assumes the transposed graph $\mathcal{G}^T$ has already been calculated. The variables n and array types $\mathcal{M}, \mathcal{T}, \mathcal{C}$ describe the number of variables, marked nodes, the nodes in topological ordering and strongly connected components, respectively. The `Vertex` type is simply an alias for the `Integer` type, hence the vertices are represented by labels, where the label $i+1$ of any $v(\overline{u})$ follows directly after its truth node $v(u)$ with label i. The main procedure of this algorithm consists of three loops which sequentially carry out the following tasks:

1. Lines (14–16): Calculate topological ordering $\mathcal{T}$ of $\mathcal{G}$.
2. Lines (17–21): Assign SCCs to all $v \in \mathcal{V}$ by visiting $\mathcal{G}^T$ in reverse topological order.
3. Lines (22–24): Check if any two nodes $v(u)$ and $v(\overline{u})$ are in the same SCC. If this is the case then we proved there can not be a solution and ϕ is UNSAT.

Therefore, this algorithm still runs in linear time since $\Theta(2 \times |\mathcal{V}| + |\mathcal{E}|)$ is still linear. And theoretically, Kosaraju-Sharir will always have to do at least $\Omega(|\mathcal{V}| + |\mathcal{E}|)$ comparisons, because any possible implementation would have to visit all nodes and all edges regardless. Summarized, this algorithm can prove useful in cases where it is unknown if a puzzle has a solution or not. The formula of the puzzle in question should, however, contain only 2-SAT clauses that completely cover all constraints for this algorithm to work, since 2-SAT can be solved in polynomial (in this case linear) time.

Algorithm 2: 2-satisfiable Kosaraju-Sharir

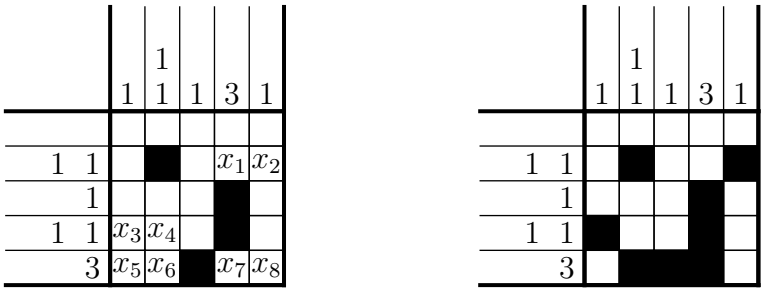
```
Graph : G ≝ ⟨V, E⟩, G^T ≝ ⟨V^T, E^T⟩
Integer : n
Boolean → Vertex : M
Integer → Vertex : T, C

Data: Vertex : x
Procedure OrderDFS
    M_x ← True                                  ▷ Vertex has now been visited
    for u ∈ {y | [x → y] ∈ E} do
        if ¬M_u then
            OrderDFS(u)                         ▷ Visit neighbours
    T_nextundef ← x                             ▷ Next undefined value (append x)

Data: Vertex : x, Integer : c
Procedure AssignDFS
    C_x ← c                                     ▷ Set SCC of x to c
    for u ∈ {y | [x → y] ∈ E^T} do
        if C_u = Undefined then
            AssignDFS(u, c)                     ▷ Assign reachable vertices to same SCC

Result: Boolean : ∃β: X ↦ {True, False} | φ(N) = True
begin
    ∀x ∈ {M_1, ..., M_n} x ← False
    for i ← 0 to n do
        if ¬M_i then
            OrderDFS(i)
    for i ← 0 to n ∧ j = 0 do
        Vertex : v ← T_{n−i−1}                  ▷ Reverse topological order
        if C_v = undefined then
            AssignDFS(v, j)
            j ← j + 1
    for i ← 0 to n do
        if C_{2i} = C_{2i+1} then
            return false                        ▷ c[i] = v(u) and c[i+1] = v(ū)
    return true
```

3.2.4 The 2-SATSolver algorithm

In the previous subchapter we introduced an algorithm to determine *if* there exists a solution for a partially solved puzzle $\mathcal{N}$ with $\phi(\mathcal{N})$ in 2-CNF. Now we will discuss an actual solving strategy for Nonograms based on 2-satisfiability, the 2-SATSolver. As mentioned before, we can derive the implicative relations between pixel values quite easily to construct the implication graph $\mathcal{G}(\phi)$ with $2n$ nodes, n being the number of unknown pixels in $\mathcal{N}$. Say we come across a path $v(x) \to v(\overline{x})$ or $v(\overline{x}) \to v(x)$, starting from $v(y)$ for some unknown pixels x, y (including $x = y$), and we assume there exists a truth-assignment for ϕ (hence ϕ is a yes-instance of Kosaraju-Sharir). Then there is no doubt that we have found a contradiction for x or $\overline{x}$. Again, in a similar way as the SimpleSolver strategy, this approach can only determine fixed pixel values since contradictions are found for instances for which we can logically proof they are incorrect. Therefore, the puzzles that this algorithm can solve are all uniquely solvable as well, since there is no guessing involved in the process. To illustrate how 2-SATSolver operates, let us take a look at the following example in Figure 5.

Example 3.1.

Let $\mathcal{N}$ be a partially solved Nonogram that SimpleSolver got stuck on, ϕ its derived formula such that $\phi = \bigwedge_{[\varphi \to \psi] \in \mathcal{E}} \neg\varphi \vee \psi$ with $\mathcal{G}(\phi) = \langle \mathcal{V}, \mathcal{E} \rangle$ being the implication graph constructed from ϕ.



$$\mathcal{G}(\phi) = \Big\langle \underbrace{\{v(x_1), v(\overline{x}_1), \ldots, v(x_8), v(\overline{x}_8)\}}_{2n \text{ vertices}}, \underbrace{\{[x_1 \to \overline{x}_2], \ldots, [\overline{x}_8 \to x_6]\}}_{0 \leq i \leq |\mathcal{V}|^2 \text{ edges}} \Big\rangle$$

Figure 5: Uniquely solvable 5×5 Nonogram with unique solution and implication graph definition.

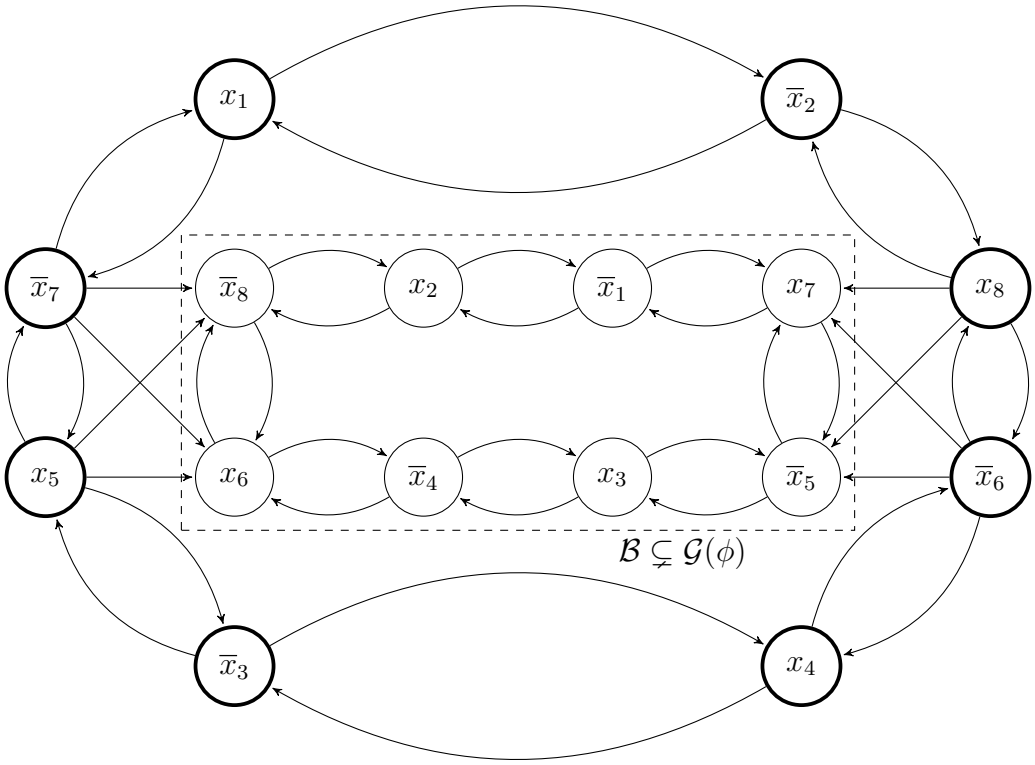


Figure 6: Implication graph of Nonogram in Figure 5 with only 2 edge crossings.

The implication graph in Figure 6 gives us a clear image of the topological problem. This digraph can be separated into two strongly connected components, namely $\mathcal{A}$ and $\mathcal{B}$, the nodes with and without thick borders around them, respectively. Notice that

$$\forall v \in \mathcal{V}\ (((v(x) \in \mathcal{A}) \iff (v(\overline{x}) \in \mathcal{B})) \wedge ((v(\overline{x}) \in \mathcal{A}) \iff (v(x) \in \mathcal{B})))$$

And because for any node $v \in \mathcal{B}$ it holds that there is no transition to an outer ring node, but all outer ring nodes can reach every node in $\mathcal{B}$, $\mathcal{B}$ gets completely isolated from $\mathcal{A}$. Therefore, any node in $\mathcal{B}$ can be reached from a starting point in the outer ring $\mathcal{A}$, but not the other way around. We could, for example, completely walk the outer ring and then enter $\mathcal{B}$ at any of the 4 entry nodes. Then, we would find contradictions for all outer ring nodes and conclude that we have found a mapping satisfying ϕ, which in this case would be either $f\colon x \to (v(x) \notin \mathcal{A})$ or $f\colon x \to (v(x) \in \mathcal{B})$ or some other expression that is logically equivalent. In this specific example, we would be done and the unique solution for the puzzle can be obtained by mapping `True` $\to$ ■ and `False` $\to$ □ for the found truth-assignment.

2-SATSolver, however, does not make use of this approach where the strongly connected components are being determined. For finding contradictions, it is sufficient to check for all nodes, normal and negated, if a contradiction can be found on any possible path in the digraph. If this is the case, we fill in our newly found pixel value and halt 2-SATSolver to run SimpleSolver. This is done to relieve the intensity of the algorithmic procedure, since 2-SATSolver is computationally more expensive than SimpleSolver. The 2-SATSolver is therefore an assembly of 2-SAT procedures and the SimpleSolver algorithm. After fixing a certain pixel that we want to investigate, contradictions are found by monitoring a cross table (i.e., see Figure 7), consisting of all nodes and corresponding Boolean values whether they have been visited or not. Once a new node has been entered in the table, it will be queued to have all its implications checked as well, effectively applying this procedure on all its neighbours and so on and so on. In Chapter 3.3 we combine both forces of the SimpleSolver and 2-SATSolver in making an extended version of our algorithm based on 2-satisfiability.

Contradiction table for $v(\overline{x}_{i+1}) = \texttt{True}$			
Nodes $v \in \mathcal{G},\ i = 0$	Marked for finished	Has been visited	Contradiction found
$v(x_i)$	`False`	`True`	`False`
$v(\overline{x}_i)$	`True`	`False`	`False`
$v(x_{i+1})$	`True`	`True`	`False`
$v(\overline{x}_{i+1})$	`True`	`True` (fixed)	`True`[1]
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v(x_{2n})$	`False`	`True`	`False`
$v(\overline{x}_{2n})$	`False`	`False`	`False`

Figure 7: Example cross table for 2-SATSolver.

3.2.5 FourSolver, a case study

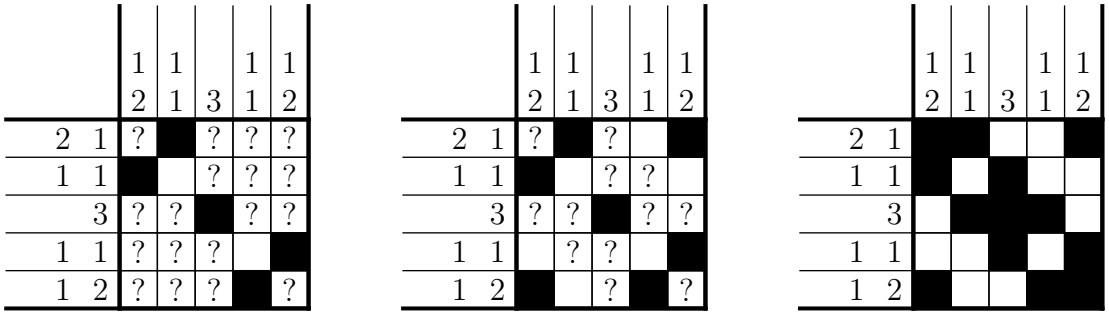
In [BK12], Batenburg et al. propose a (p, q)-Solver. This solving strategy takes into account $p \cdot q$ intersections (pixels) at a time, taken from selecting p rows and q columns. In a regular $m \times n$ puzzle, this gives $\sigma = \binom{m}{p} \times \binom{n}{q}$ possible intersection combinations per image. The concept of (p, q)-*hardness* is introduced (puzzles solvable by this method), where the $(2, 2)$-Solver is referred to as the FourSolver algorithm. As the name

[1] Contradicting node $v(x_{i+1})$ has been visited, and therefore we conclude $\neg v(\overline{x}_{i+1})$

suggests, only 4 pixels are assessed for finding contradictions. Therefore, the amount of observable intersection combinations is reduced to $\frac{1}{4} \times \frac{m! \cdot n!}{(m-2)! \cdot (n-2)!}$. Instead of using an implication graph with transitions, the procedure constructs a lookup table at the beginning of the puzzle by simply deducing contradictions from the line descriptions. Intuitively, this approach is far more efficient than 2-SATSolver, but not as complete. In fact, having a lookup mechanism for every single quadruplet of pixels is equivalent to creating nothing more than σ implication graphs over the pixel intersection combinations. Implications between these smaller graphs themselves are left out, and therefore, there is no chance of finding such an inter-quadruplet contradiction. Hence both algorithms would behave the same way for 2×2 or smaller puzzles, because FourSolver would be assessing the entire puzzle, just like 2-SATSolver always does regardless of the puzzle size. To better understand FourSolver's shortcomings, take a look at the following example.

Example 3.2.

Let $\mathcal{N}$ be a partially solved Nonogram that SimpleSolver got stuck on, and 4-Solver the algorithm defined in [BK12], slightly modified so that it constructs subgraphs $\mathcal{G}_\Delta \subseteq \mathcal{G}$ of size $|\mathcal{G}_\Delta| \leq 4 \times 2$ instead of using a lookup-table.



(a) SimpleSolver **(b)** 4-solver **(c)** 2-SATSolver

Figure 8: Progress made by every solver on a uniquely solvable puzzle.

As can be seen in Figure 8, 4-SOLVER is unable to solve the entire puzzle, but it makes some progress even on top of SIMPLESOLVER, which only finds the middle pixel and the formations in the upper left and lower right corners (see Figure 8 (c)). 4-SOLVER, for example, finds a contradiction in $r = (0, 2)$ and $c = (0, 3)$, when counting rows and columns starting from 0. For this specific instance, the algorithm yields the digraph in Figure 9.

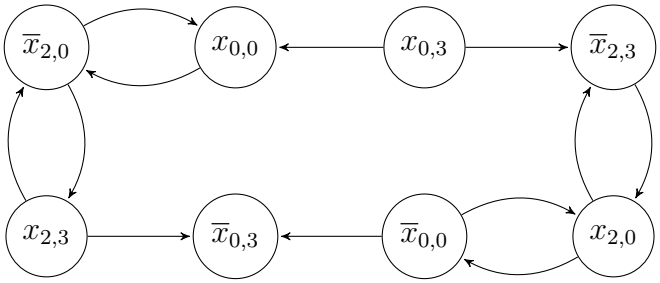


Figure 9: 4-SOLVER constraint problem for $r = (0, 2)$ and $c = (0, 3)$.

Only for node $x_{0,3}$ we can find a contradiction by choosing any of the paths to $\overline{x}_{0,3}$ via the right or left side of $\mathcal{G}_\Delta$. Hence we can conclude that the corresponding value of the intersecting pixel of $r = 0,\ c = 3$ is $\square$. Notice that, because of the symmetry in the line descriptions, an equivalent digraph would be constructed for $r = (0, 3)$ and $c = (0, 2)$. This graph would therefore have a path $x_{4,1} \to \overline{x}_{4,1}$ just like the graph in Figure 9 has a path $x_{0,3} \to \overline{x}_{0,3}$. After running SIMPLESOLVER, that finds some empty pixels, 4-SOLVER cannot find anything and this gives us the result in Figure 8 (a).

This brings us to the 2-SATSOLVER, that proves to be more useful in solving this puzzle. It effectively finds a superset of the paths that 4-SOLVER finds. In general, the distinct pixels 4-SOLVER can find on a path is limited to 4, and therefore, usually less than the $m \cdot n$ of 2-SATSOLVER. For instance, the path

$$\underline{x_{3,1}} \to \overline{x}_{2,1} \to x_{2,4} \to \overline{x}_{4,4} \to x_{4,0} \to x_{3,0} \to \underline{\overline{x}_{3,1}}$$

could never be found by 4-SOLVER, as it features 6 different pixels, but is crucial to the solution of the puzzle. Again, because of the symmetry, a path from $\overline{x}_{1,3}$ to $x_{3,1}$ is found as well. It seems obvious that if a puzzle is $(2, 2)$-hard, it is also 2-SAT-hard. This case study, however, proves by contradiction that the reverse is not true. But, as the experiments in Chapter 4 will reveal, every puzzle that is $(2, 2)$-hard is also 2-SAT-hard,

at least for the small and medium size puzzles tested. In this Chapter 4, where we classify the puzzles, the FourSolver is left aside on purpose since it is simply an incomplete version of 2-SATSolver and because Batenburg et al. already performed experiments with FourSolver on medium and large sized puzzles in [BK12].

3.3 Extended 2-SAT solving

In this subchapter we briefly discuss a problem of the 2-SATSolver as introduced in Chapter 3.2.4. We propose an improvement of the algorithm and upscale it to the 2S-Solver, and work out an example where the original algorithm fails but the new one solves the puzzle.

3.3.1 Problem description

In Chapter 3.2.4 we discussed an algorithm capable of finding fixed pixel values in an implication graph using a contradiction table. The 2-SATSolver makes iterations through the nodes of the implication graph, and applies the cross table procedure explained in Chapter 3.2.4 to every node. If some contradiction is found then the corresponding pixel value of this node is set to its definitive value and the procedure is halted and repeated again after applying SimpleSolver by constructing the new digraph if the puzzle is not solved.

Even though it may look like 2-SATSolver is in every way superior to the SimpleSolver algorithm, it turns out be rather different. Recall that 2-SATSolver halts if the contradiction table has been filled as far as possible and no contradiction has been found. The 2-SAT procedure will end, the algorithm applies SimpleSolver, and if nothing is found then we will stop searching for new contradictions. Because we evaluate the relations of single nodes, some crucial, yet extremely obvious information is lost and neglected. The 2-SATSolver might halt at some point after fixing a pixel a certain value where the relations hit a dead end, but in the process of entering values in the cross table, we have created a situation where we assigned (possibly wrong) virtual values to some pixels in some row or column where SimpleSolver can actually progress. The 2-SATSolver algorithm seems unable to deduce these simple relations because they feature the values of multiple nodes combined. We discovered this problem while examining the remainder of the uniquely solvable puzzles that 2-SATSolver is unable to solve. This is a relatively small set of puzzles. The exact numbers can be found in the experiments Chapter 4.

3.3.2 A simple solution

A straightforward solution has been implemented on top of the original 2-SATSolver. While adding values to the table, we monitor an extra virtual puzzle grid and run SimpleSolver if we hit any dead ends, in effect rescuing the 2-satisfiability procedure where the original algorithm would stop if no new values would have been found. We then enter any new values the procedure found into the table, and therefore they will be queued to be checked out. This way, we get the best of both worlds as we use SimpleSolver not only to gradually progress on the puzzle as intended, but also to shed new light on situations where the conventional 2-satisfiability approach is inferior. In Figure 10 we take a look at the a difficult puzzle, which also happens to be the hardest 5×5 2S-hard Nonogram.

Example 3.3.

Let $\mathcal{N}$ be a partially solved Nonogram that both SimpleSolver and 2-SATSolver got stuck on and ■ and □ denote the black and white pixels values following from 2-SAT implications after trying a certain pixel.

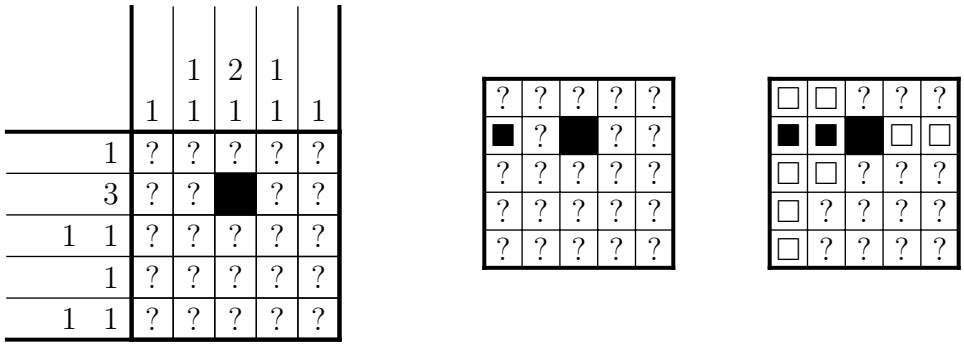


Figure 10: SimpleSolver progress (left panel) including 2-SATSolver progress (none). After trying the black pixel (right panel) we find no contradiction and stop searching for $(0, 1) = ■$. Notice that, in the third row ($r = 2$), SimpleSolver could make some progress on this assumption.

In the right panel of Figure 10, it is entirely obvious what the solution to the third row should be. 2-SATSolver, however, is unable to deduce this from the formula and stops searching for this pixel. If we let SimpleSolver interfere at this point (just like 2S-Solver would do), we get the following solving sequence.

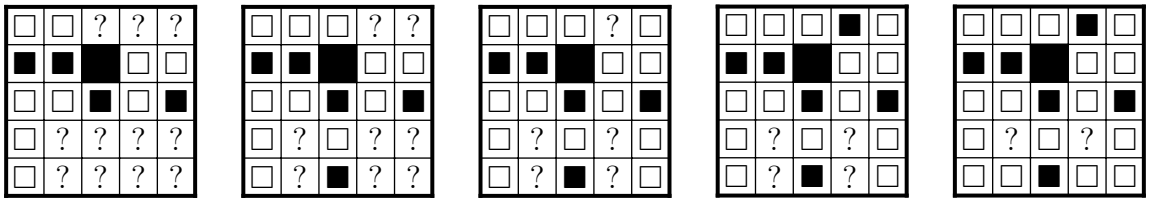


Figure 11: First row fill by SimpleSolver (first panel); middle and right column fill by either SimpleSolver or 2-satisfiability on middle pixels (second and third panel); upper row fill by SimpleSolver (fourth panel) and bottom row fill by either SimpleSolver or 2-SATSolver.

At this point in time, if we follow the column descriptions and set the two remaining unknown pixel values to black, we would end up with a contradiction in row 4 ($r = 3$). Therefore, the assumption that we started with in Figure 10 is wrong, hence $(1, 0)$ should be white. Knowing this, we derive three other pixel values from creating a new graph and continue with assuming pixel $(3, 2) = \square$. Note that choosing the assumptions in a different order might result in less or more work to do. From backtracking, we know that this specific puzzle needs at least 2 confirmed assumptions by 2-SAT (or correct guesses) to become simple-solvable.

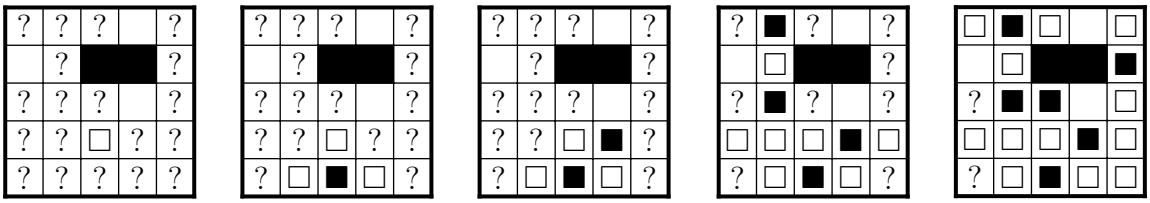


Figure 12: Assumption $(3, 2) = \square$ (first panel); obvious 2-SAT implications (second panel); SimpleSolver on fourth column (third panel); 2-SATSolver or SimpleSolver on fourth row and SimpleSolver on second column (fourth panel); 2-SATSolver or SimpleSolver on second row; fifth column and first row (fifth panel), respectively.

In the final panel of Figure 12, a contradiction appeared in the third row. The line description $\mathcal{D}_3 = (1, 1)$ enforces any node from this row to have an implication of the form $v(x_{2,i}) \to v(\overline{x}_{2,i+1})$ for $i < m - 1$ and $v(x_{2,i}) \to v(\overline{x}_{2,i-1})$ for $i \in \{1, 2, 3, 4\}$. Hence both nodes would find a such a path to each other and we conclude that $(3, 2) = \blacksquare$. After this conclusion, SimpleSolver is able to fully solve the puzzle, but it still takes a considerable amount of 6 sweeps to solve it. As said before, this is a special nonsimple 2S-hard puzzle that 2-SATSolver is unable to solve on its own. It is the only 5×5 puzzle out of a small group of uniquely solvable not-2-SAT-hard puzzles that takes at least 2 assumptions to get solved by SimpleSolver afterwards. For the other puzzles, 1 assumption suffices.

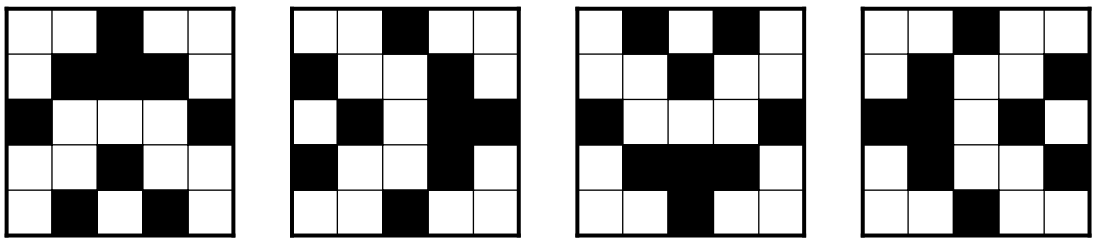


Figure 13: The four rotations of the 2S-hard puzzle dealt with in Example 3.3.2.

4 Classification

In this chapter we report and discuss the outcome of the experiments performed on all puzzles of specific sizes. First we analyze the relatively small sets of 3×3 up to 4×4 puzzles. We then continue with medium size Nonograms, being the 4×5, 5×5 and 5×6 puzzles. Because of the immensely large image space of 6×6 puzzles ($2^{6 \cdot 6} = 68,719,476,736$ images), we analyze a sample taken from the images in the large puzzle section. All experiments on small and medium size Nonograms have been run on a 64bit Ubuntu 16.04 system with an Intel i7-6700HQ overclocked @4GHz with 8 logical cores. Experiments on large puzzles have been performed on several[2] machines of the LIACS Data Science Lab [lab] simultaneously, but most work has been done by the 64bit CentOS Linux 7 "Mithril" workstation with 128 logical cores (Intel Xeon E5-4667 v3 @2500MHz).

[2]Miraculously, these machines became very popular and therefore more and more occupied near the end of this term.

4.1 Small size Nonograms

The Nonograms in the uniquely solvable sets of these puzzles are rather uncomplicated. For 3×3 puzzles, the SimpleSolver even solves all uniquely solvable Nonograms. The remaining 128 puzzles have multiple solutions. As from 3×4 and higher, some uniquely solvable puzzles turn out to be nonsimple. The results for other solvers on those puzzles can be found on the next page in Figure 15. What is interesting is the relative drop in the number of simple puzzles of 3×5 Nonograms, which has an image space half the size of the 4×4 ($2^{3 \cdot 5} = 32,768$ vs. $2^{4 \cdot 4} = 65,536$). This drop could be due to the larger row width of 3×5 puzzles, that introduces more possibilities and therefore also more difficulties for harder line descriptions like (1) and $(1, 1)$.

Difficulty	simple 3×3 Nonograms	simple 3×4 Nonograms	simple 3×5 Nonograms	simple 4×4 Nonograms
0	0	0	0	0
0.5	0	0	0	0
1	6	9	17	16
1.5	42	127	425	480
2	236	1,240	7,491	10,666
2.5	76	900	6,855	11,528
3	24	412	4,088	10,916
3.5	0	284	2,914	7,856
4	0	80	988	3,036
4.5	0	56	638	4,616
5	0	0	256	616
5.5	0	8	204	864
6	0	0	0	120
6.5	0	0	176	440
7	0	0	0	0
7.5	0	0	0	56
8	0	0	0	0
8.5	0	0	0	24
9	0	0	0	0
Total	384	3,116	24,052	51,234
% of $2^{m \cdot n}$	75.00%	76.07%	73.40%	78.18%

Figure 14: Results for small size simple Nonograms.

Difficulty	2-SAT-hard 3×3	2-SAT-hard 3×4	2-SAT-hard 3×5	2-SAT-hard 4×4
1	0	36	178	1,128
2	0	0	0	0
Total	0	36	178	1,128
% of $2^{m \cdot n}$	0.00%	0.88%	0.54%	1.72%
Difficulty	2S-hard 3×3	2S-hard 3×4	2S-hard 3×5	2S-hard 4×4
1	0	0	0	0
Total	0	0	0	0
% of $2^{m \cdot n}$	0.00%	0.00%	0.00%	0.00%
Difficulty	2S progress 3×3	2S progress 3×4	2S progress 3×5	2S progress 4×4
1	0	44	456	2,304
2	0	8	162	356
Total	0	52	618	2,660
% of $2^{m \cdot n}$	0.00%	1.27%	1.89%	4.06%
Difficulty	2S no progress 3×3	2S no progress 3×4	2S no progress 3×5	2S no progress 4×4
1	122	856	7,484	10,156
2	6	36	436	334
3	0	0	0	24
Total	128	892	7,920	10,514
% of $2^{m \cdot n}$	25.00%	21.78%	24.17%	16.04%

Figure 15: Results for small size nonsimple Nonograms.

No uniquely solvable Nonograms in this size category are so hard that 2S-Solver has to interfere. The 2-SATSolver successfully solves *all* uniquely solvable Nonograms for the sizes in the table in Figure 15. Therefore, there are no 2S-hard Nonograms that are not 2-SAT-hard, yet. 2S-Solver can, however, make some progress on the remaining nonuniquely solvable puzzles. This is also true for 2-SATSolver, but to reduce the overload of information and since 2S-Solver is just an extension of 2-SATSolver, we only show the progress of 2S-Solver. As difficulty measure for nonsimple puzzles, we use the minimal amount of clues (or correct assumptions) required to fully solve the puzzle with SimpleSolver. This difficulty measure shall hereon be used throughout this chapter for nonsimple puzzles.

4.2 Medium size Nonograms

Difficulty	simple Nonograms	Difficulty	simple Nonograms
0	0	6.5	14,884
0.5	0	7	1,076
1	36	7.5	5,544
1.5	2,248	8	108
2	107,474	8.5	2,496
2.5	139,184	9	8
3	186,274	9.5	844
3.5	135,348	10	0
4	69,428	10.5	380
4.5	82,404	11	0
5	21,608	11.5	48
5.5	27,576	12	0
6	4,820	12.5	44
		Total	801,832 (76.47%)

Figure 16: Results for simple 4×5 Nonograms.

Difficulty	2-SAT-hard Nonograms	2S-hard Nonograms	some progress by 2S-Solver	no progress by 2S-Solver
1	12,796	0	40,388	173,626
2	4	0	11,522	7,944
3	0	0	96	368
Total	12,800	0	52,006	181,938
% of $2^{m \cdot n}$	1.22%	0.00%	4.96%	17.35%

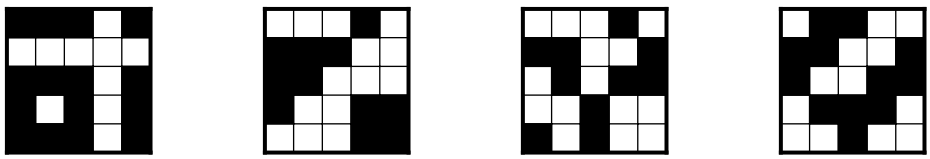
Figure 17: Results for nonsimple 4×5 Nonograms.

Again, as with previous puzzle sizes, the 2-SATSolver solves all uniquely solvable Nonograms of this size. Around 22% of the nonuniquely solvable puzzles made some progress after applying 2S-Solver. We now quickly move on to the 5×5 puzzles where we discover our first 2S-hard puzzles.

Difficulty	simple Nonograms	Difficulty	simple Nonograms
0	0	8.5	160,492
0.5	0	9	3,016
1	108	9.5	63,832
1.5	15,336	10	504
2	1,932,817	10.5	30,328
2.5	2,938,878	11	16
3	5,829,718	11.5	9,584
3.5	4,135,740	12	0
4	3,042,068	12.5	4,680
4.5	2,808,316	13	0
5	1,171,530	13.5	1,048
5.5	1,314,148	14	0
6	322,468	14.5	408
6.5	741,316	15	0
7	92,336	15.5	24
7.5	340,812	16	0
8	16,980	16.5	8
		Total	24,976,511 (74.44%)

Figure 18: Results for simple 5×5 Nonograms.

The total amount of uniquely solvable 5×5 Nonograms is $25,309,575$ out of a total of $2^{5 \cdot 5} = 33,554,432$ puzzles of this size. Hence 333,064 uniquely solvable puzzles can not be solved by SimpleSolver. 8,577,921 (25.56%) of the puzzles remain unsolved. The vast majority of the simple puzzles (92.78%) have a difficulty $2 \leq x \leq 5.5$, an interval that spans only 25% of the difficulty domain (excluding 0 and 0.5).



(a) **(b)** **(c)** **(d)**

Figure 19: A very simple Nonogram of difficulty 1 (a), a reasonably difficult Nonogram of difficulty 8 (b), a very hard Nonogram of difficulty 13.5 (c) and the hardest simple Nonogram of difficulty 16.5 (d).

Difficulty	2-SAT-hard Nonograms	2S-hard Nonograms	some progress by 2S-Solver	no progress by 2S-Solver
1	331,926	1,036	1,406,828	5,868,368
2	98	4	578,308	351,191
3	0	0	18,970	20,827
4	0	0	34	331
Total	332,024	1,040	2,004,140	6,240,717
% of $2^{m \cdot n}$	0.99%	0.00%	5.97%	18.60%

Figure 20: Results for nonsimple 5×5 Nonograms.

2-SATSolver manages to solve 99.67% of all nonsimple uniquely solvable puzzles. The remaining 1,040 nonsimple uniquely solvable puzzles turn out to be 2S-hard. Therefore, all 5×5 uniquely solvable puzzles have been solved by one of our three solving strategies. On roughly 2 million ambiguous puzzles (24.31% of the unsolved puzzles), 2S-Solver makes some progress, but is not designed to solve these kind of puzzles with multiple solutions. These solvers require a considerable amount of guessing, and even with some smart implementations probably still amount to plain brute forcing strategies. The progress that was made by 2S-Solver obviously only covers fixed pixels in these puzzles with multiple solutions, and not the actual switching components.

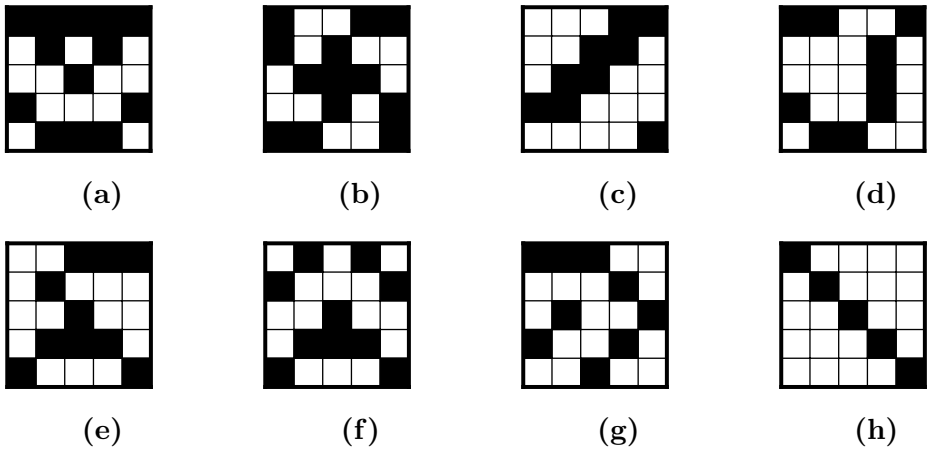


Figure 21: A happy and not so happy 2-SAT-hard Nonogram of difficulty 1 (a) and 2 (b), 2S-hard Nonogram of difficulty 1 (c), possible solutions of Nonograms where some progress was made by 2S-Solver of difficulties 2 (d), 3 (e) and 4 (f) and where no progress was made by 2S-Solver of difficulties 3 (g) and 4 (h).

Difficulty	simple Nonograms	Difficulty	simple Nonograms
0	0	11	5,044
0.5	0	11.5	878,832
1	333	12	656
1.5	105,039	12.5	405,176
2	29,597,291	13	52
2.5	64,058,939	13.5	159,192
3	154,031,378	14	0
3.5	135,235,289	14.5	66,936
4	100,482,848	15	0
4.5	108,441,043	15.5	22,676
5	43,731,472	16	0
5.5	59,634,572	16.5	8,192
6	13,806,074	17	0
6.5	33,970,080	17.5	2,044
7	4,153,728	18	0
7.5	16,599,950	18.5	688
8	952,584	19	0
8.5	8,418,054	19.5	88
9	206,736	20	0
9.5	4,023,698	20.5	52
10	36,408	21	0
10.5	1,970,216	21.5	12
		Total	781,005,372 (72.74%)

Figure 22: Results for simple 5×6 Nonograms

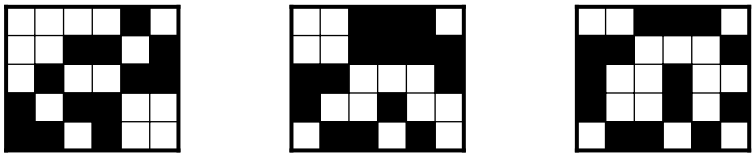


Figure 23: The three hardest simple 5×6 Nonograms of difficulty 21.5, all shown in their first rotation as they occur in lexicographical order. Note that Nonograms of size $m \times n$ where $m \neq n$ have at most 4 rotations, unlike the conventional square puzzles that can take a maximum of 8 different settings.

Difficulty	2-SAT-hard Nonograms	2S-hard Nonograms	some progress by 2S-Solver	no progress by 2S-Solver
1	13,313,573	52,378	48,115,336	189,306,574
2	6,302	708	24,438,941	14,505,776
3	0	0	1,904,822	1,011,238
4	0	0	32,718	47,884
5	0	0	44	158
Total	13,319,875	53,086	74,491,861	204,871,630
% of $2^{m \cdot n}$	1.24%	0.00%	6.94%	19.08%

Figure 24: Results for nonsimple 5×6 Nonograms.

Exactly 794,378,773 (73.98%) of the 5×6 puzzles are uniquely solvable. Hence there are 440 uniquely solvable puzzles that are not 2S-hard. So far, we have not seen any uniquely solvable nonsimple puzzles that 2S-Solver is unable to solve, but now we actually found some. Compared to 5×5 puzzles, there are relatively more nonsimple and less simple puzzles. Still, the number of 2S-hard Nonograms is extremely low and difficulty 3 for this class of puzzles has also not been reached here. A very small amount of 202, not necessarily uniquely solvable, puzzles require at least 5 clues to become simple solvable. If we look at the different tables from this chapter, the percentage of simple and uniquely solvable puzzles seems to decrease as the puzzles get larger. At the same time, the share of 2-SAT-hard, 2S-hard and progress gaining puzzles seems to increase. This will continue to go on for larger puzzles as well, probably.

4.3 Large size Nonograms

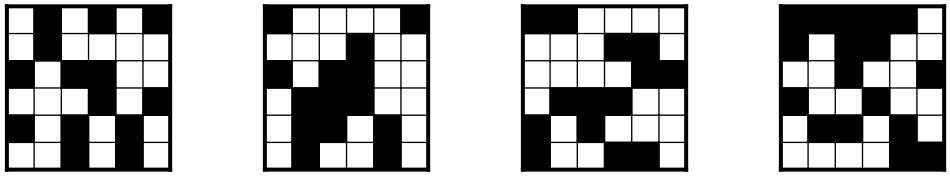
So far, we have analyzed the complete image spaces of the small and medium size puzzles dealt with in this chapter. As the puzzles increase in size, not only do their image spaces expand, but, naturally, the computational workload required to solve every individual (larger) puzzle increases as well. For instance, the image space expands from 5×6 to 6×6 puzzles by a factor of $\frac{2^{6 \cdot 6}}{2^{5 \cdot 6}} = 2^6 = 64$. The overall puzzle size increases by 20%. By cause of these facts, solving large image spaces with all three solving strategies as a whole, whilst also backtracking the difficulty measures for SimpleSolver-based difficulties, simply takes too much CPU time.

In order to still obtain a proper overview of the 6×6 puzzles we use a probability sampling method; more specifically, *systematic sampling* [Agr18]. This means dividing the population (the puzzles) in regular intervals and drawing a random sample over each subgroup. We analyze a little over $\sim$ 1% of the 68,719,476,736 6×6 puzzles. We chose the sample size to be $N = 687,194,816$ such that both the sample size and the population size $2^{6 \cdot 6}$ are divisible by 64 (for both sampling and computational purposes). The population is therefore divided in 64 different subgroups, with each group containing exactly $\frac{2^{6 \cdot 6}}{2^6} = 2^{30}$ lexicographic consecutive puzzles. Accordingly, we draw a random sample of exactly $\frac{N}{64}$ individuals over each subgroup. Even though this approach might lead to some form of sampling bias since puzzles with the same traits are not necessarily clustered together, it excludes the possibility that a certain part of puzzles are left out by default. The results for simple puzzles can be found in Figure 25.

Batenburg and Kosters also performed more complete experiments on simple 6×6 puzzles. They utilize a different difficulty measure that is similar to ours, but counts the number of sweeps required by an H-Sweep-starting SimpleSolver to fully solve the puzzle. It therefore does not take into account the sweep difference explained in Chapter 3.1. If we compare our results to these results in [BK12], we notice that the highest difficulty they observed is 28. These puzzles would be scattered over our difficulties 27.5, 28 and 28.5. The random sample we drew clearly does not include any of these puzzles, as these difficulties are not present in Figure 25. What seems to be pure coïncidence, however, is that the percentage of simple puzzles in respect to the total amount of puzzles in [BK12] is exactly same same to the percentage of simple puzzles over our random sample size (70.76%). The percentages of nonsimple puzzles in Figure 26 seem to increase compared to previous puzzle sizes. Therefore, we suspect that as puzzles increase in size, the overall share of simple puzzles decreases while more and more puzzles are classified as 2-SAT-hard and 2S-hard.

Difficulty	simple Nonograms	Difficulty	simple Nonograms
0	0	13	271
0.5	0	13.5	265,790
1	14	14	32
1.5	10,517	14.5	130,616
2	7,668,185	15	3
2.5	22,502,367	15.5	54,215
3	75,039,981	16	0
3.5	77,620,188	16.5	2,5236
4	66,586,358	17	0
4.5	75,366,690	17.5	9,732
5	33,802,328	18	0
5.5	47,531,148	18.5	4,068
6	12,649,365	19	0
6.5	29,380,115	19.5	1,469
7	4,209,214	20	0
7.5	15,103,012	20.5	526
8	1,173,952	21	0
8.5	8,469,436	21.5	199
9	296,300	22	0
9.5	4,266,482	22.5	56
10	65,063	23	0
10.5	2,326,222	23.5	19
11	12,299	24	0
11.5	1,121,113	24.5	8
12	1,946	25	0
12.5	591,483	25.5	2
		Total	486,286,020 (70.76%)

Figure 25: Results for simple 6×6 Nonograms (percentage taken over N).



(a) **(b)** **(c)** **(d)**

Figure 26: Four random 6×6 2S-hard Nonograms in lexicographic order.

Difficulty	2-SAT-hard Nonograms	2S-hard Nonograms	some progress by 2S-Solver	no progress by 2S-Solver
1	11,139,168	48,830	33,151,702	122,291,457
2	10,019	969	19,871,815	10,857,209
3	0	0	2,505,368	836,406
4	0	0	119,072	72,680
5	0	0	1,502	2,577
6	0	0	7	15
Total	11,149,187	49,799	55,649,466	134,060,344
% of N	1.62%	0.01%	8.10%	19.51%

Figure 27: Results for nonsimple 6×6 Nonograms (percentages taken over N).

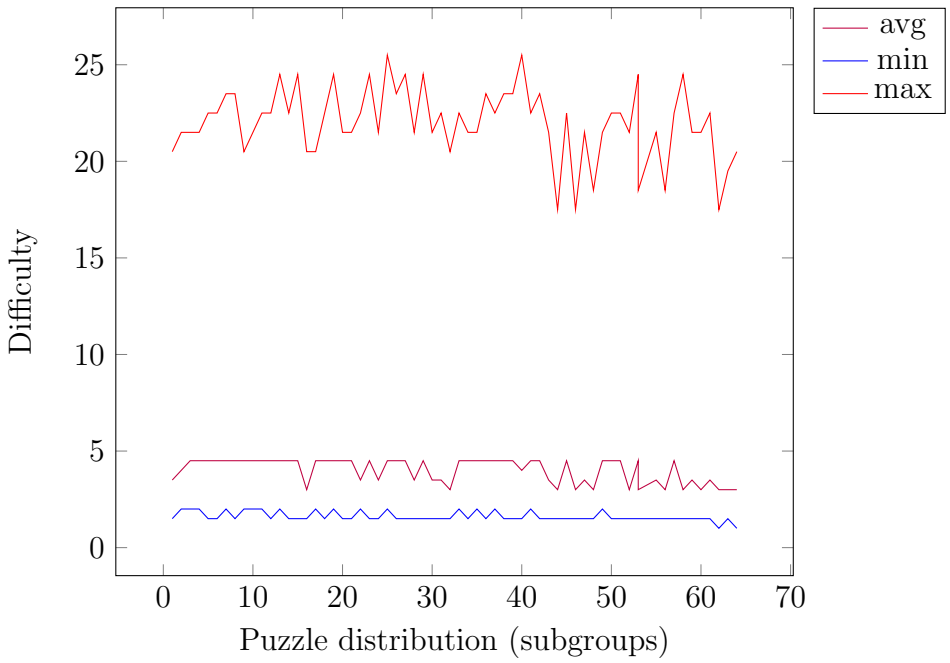


Figure 28: Results for each individual subgroup in the random sample for simple 6×6 puzzles. The average, minimum and maximum difficulty observed are plotted for the puzzle distribution, where the x-ticks are in lexicographic order.

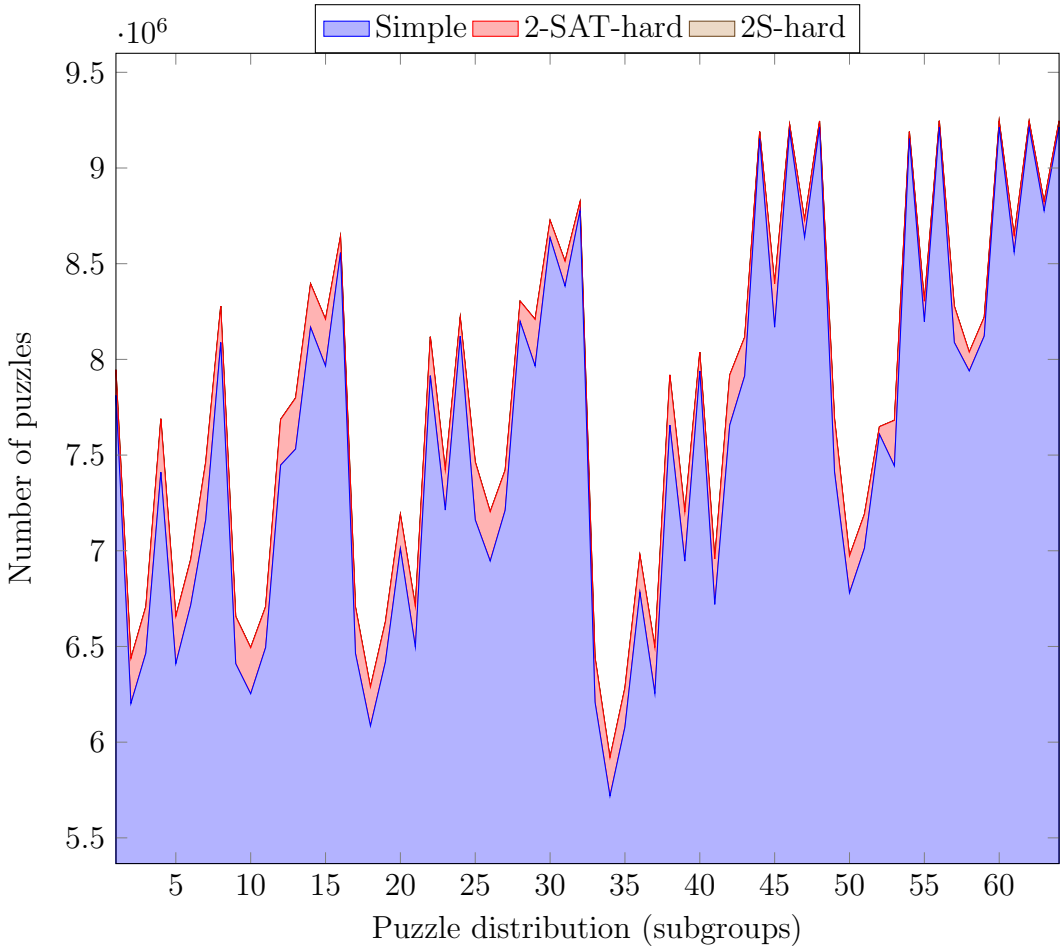


Figure 29: Solver distributions for the individual subgroups in the random sample. Subgroups are in lexicographical order. Some kind of pattern is recognizable in the plot and has to with the order in which the puzzles are generated. The 2S-hard puzzles are completely overrun by the other two solving strategies and cannot be clearly distinguished in this figure.

5 Conclusions and Further Research

In this thesis we proposed three different solving strategies for Nonogram puzzles. We began by introducing the SimpleSolver in Chapter 3.1. It turns out that the majority of the puzzles can be solved by this algorithm based on logic reasoning steps. For the remaining unsolved uniquely solvable puzzles, we proposed the 2-SATSolver. By alternating a 2-satisfiability procedure with the SimpleSolver algorithm, we discovered the group of 2-SAT-hard puzzles. Able to solve all uniquely solvable puzzles for smaller size variants, we found a relatively small amount of uniquely solvable puzzles that are not 2-SAT-hard in experiments on medium size Nonograms in Chapter 4.2. By integrating the SimpleSolver in the 2-satisfiability procedure of the 2-SATSolver we were able to solve these remaining 5×5 puzzles, classifying them as 2S-hard. Despite this improvement, for 5×6 and large size Nonograms, a portion of the uniquely solvable puzzles remain unsolved.

It is difficult to say something about the characteristics of more difficult puzzles. The number of guesses required to solve the puzzle by a procedure of simple logical reasoning steps seems an appropriate difficulty measure, as every guess or assumption features a series of new implications that might lead to either nothing or a contradiction. As the puzzler guessed "wrong", he or she wasted time by computing and filling in all pixel relations. Reducing the amount of required guesses naturally decreases the chance of guessing the wrong value for a pixel. However, other than the fact that the puzzle lines should not be too full or completely empty (easy fixing), not much can be said about how such a difficult puzzle actually might look like.

Future work should include researching both larger sized puzzles and the portion of unsolved uniquely solvable puzzles. Distributing the computational workload even further than done in this thesis might bring forth new opportunities to examine the entire image spaces of 6×6 or even larger puzzles. Since solving Nonograms is a $\mathcal{NP}$-complete problem, it is extremely likely that not all (uniquely solvable) puzzles can be solved by algorithms that run in polynomial time. However, a thorough analysis of the remaining unsolved puzzles might lead to the discovery of some new solving patterns, which in turn might conduce to the pursuit of solving and classifying as many puzzles as possible.

References

[Agr18] A. Agresti. *Statistical Methods for the Social Sciences*. Pearson Education Ltd., 2018.

[BFO+10] H. Böckenhauer, M. Forišek, J. Oravec, B. Steffen, K. Steinhöfel, and M. Steinová. The Uniform Minimum-ones 2SAT problem and its Application to Haplotype Classification. *RAIRO — Theoretical Informatics and Applications*, 44:363 – 377, 2010.

[BHKP09] K.J. Batenburg, S. Henstra, W.A. Kosters, and W.J. Palenstijn. Constructing Simple Nonograms of Varying Difficulty. *Pure Mathematics and Applications*, 20, 2009.

[BK12] K.J. Batenburg and W.A. Kosters. On the Difficulty of Nonograms. *ICGA journal*, 35:195–205, 2012.

[Car86] L. Carroll. *The Game of Logic*. Macmillan and Co., London, 1886.

[lab] LIACS Data Science Lab. http://rel.liacs.nl/dslab. Accessed: 30-6-2019, (link only internally available).

[OGSSLM+07] E. Ortiz-Garca, S. Salcedo-Sanz, J. Leiva-Murillo, A.M. Prez-Bellido, and A. Portilla-Figueras. Automated Generation and Visualization of picture-logic Puzzles. *Computers & Graphics*, 31:750–760, 2007.

[Sha81] M. Sharir. A Strong-connectivity Algorithm and its Applications in Data Flow Analysis. *Computers & Mathematics With Applications*, 7:67–72, 1981.

[SSOGPB+07] S. Salcedo-Sanz, E. Ortiz-Garca, A.M. Prez-Bellido, A. Portilla-Figueras, and X. Yao. Solving Japanese Puzzles with Heuristics. pages 224–231, 2007.

[YLC11] C. Yu, H. Lee, and L. Chen. An Efficient Algorithm for Solving Nonograms. *Appl. Intell.*, 35:18–31, 2011.