Universiteit Leiden

Opleiding Informatica

From Image to Nonogram:

Construction, Quality and Switching Graphs

Sjoerd Jan Henstra

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ABSTRACT

Nonograms are logic puzzles in which a black and white image must be reconstructed from two orthogonal projections. These projections consist of the size and order of groups of black pixels in each line.

In the first part of this thesis we describe a two step method to automatically construct Nonograms. First, a grayscale image is turned into a black and white image. Second, the black and white image is repeatedly modified until it is a uniquely solvable Nonogram. Various options for both steps are presented. We also describe ways to generate multiple Nonograms of varying difficulty based on the same grayscale image.

In the second part of this thesis we study Nonograms with multiple solutions. We compare Nonograms to a more general discrete tomography problem and determine how different solutions of the same Nonogram can be transformed into one another by repeatedly applying simple switching components. For some small examples, we describe generalized switching components which relate to Nonograms as simple switching components do to the discrete tomography problem. We present one possible measure of the complexity of a Nonogram, based on the number of solutions and the way switching components can be used to transform one solution into another.

ACKNOWLEDGEMENTS

Contents

Chapter 1

Introduction

A *Nonogram* is a logic puzzle where players try to reconstruct a black and white image from row and column descriptions. These descriptions indicate how many black cells there are in a given line (either a row or a column) and how these cells are grouped together. For example, the description $(2, 1)$ means a line contains a group of two black cells, followed by one black cell. These groups must be separated by at least one white cell. The first group may be preceded by any number of white cells, or none at all. Likewise, the second group may be followed by any number of white cells, or none at all.

Figure 1.1 shows that there are three lines of length five which fit this description. Note that the second cell from the left is black in all lines that fit the given description. Information gained this way from rows can be used to solve columns and vice versa, allowing us to solve the Nonogram in Figure 1.2 even though none of the lines can be solved on their own.

Figure 1.3 shows an example of a Nonogram with multiple solutions. Such Nonograms are generally not published as logic puzzles are generally assumed to have one correct solution.
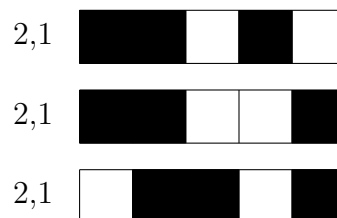

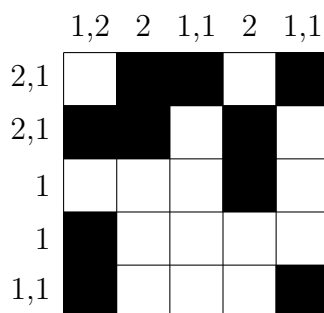Figure 1.1: Three lines which fit the same description

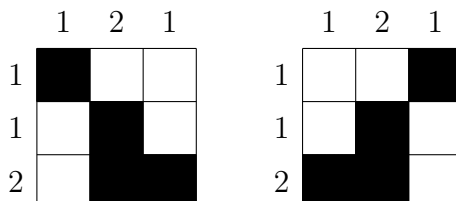Figure 1.2: Example of a Nonogram with a unique solution

Figure 1.3: Example of a Nonogram with two solutions

Nonograms became very popular in the 1990s and continue to be published frequently to this day. Constructing a large number of puzzles by hand for weekly or monthly publication takes a lot of time and constructing large Nonograms is especially difficult. The aim is to construct a puzzle that has a single, unique solution, that is not too easy or difficult to solve for humans and whose result is an image that can be recognized by humans. If one wishes to automate Nonogram construction, it makes sense to start the process with some kind of drawing or photograph. This way, there is some hope of generating a puzzle whose result is somewhat recognizable.

However, simply converting an image to black and white is not enough, as the other constraints are unlikely to be met this way. Thus, we try to create an algorithm that generates Nonograms from images. The goal is to produce Nonograms that are uniquely solvable and that resemble the input image as much as possible. We also need some measure of a puzzle's difficulty, to determine whether the puzzle is suitable for human players, and some control over how difficult the generated puzzles will be.

In addition to constructing Nonograms, we also investigate Nonograms with multiple solutions and how these solutions can be transformed into one another using switching components. Switching components are collections of cells whose values can be colored in different ways while still producing the same Nonogram projections. In particular, we well look at simple switches, collections of four cells with two possible colorings, and how simple switches can be combined into generalized switching components. We also describe a type of graph, containing all solutions of a given Nonogram, which can be used as a measure of the complexity of that Nonogram.

Chapter 2 contains some definitions used throughout this paper. The constructing of Nonograms from grayscale images is covered in Chapter 3, with Sections 3.1 through 3.4 discussing the general method and its various stages. Section 3.5 discusses several variations on the general method and Section 3.6 lists all the parameters available in the final program. The results for the construction of Nonograms are found in Section 3.7. Parts of Chapter 3 have been published in [BHKP09].

Chapter 4 covers the application of switching components to Nonograms. Section 4.1 is about distances between Nonograms and how they can be calculated. Generalized switching components are discussed in Section 4.2, Section 4.3 contains results related to switching components. and in Section 4.4 we use those results to define a measure of a Nonogram's complexity.

Finally, Chapter 5 contains conclusions and possibilities for future work.

Chapter 2

Definitions

A *Nonogram* is a set of descriptions that describe a black and white image. We define such an image I as an m by n matrix containing the values 0 and 1, with 0 denoting a white cell and 1 a black cell. A Nonogram which describes such an image consists of row descriptions and column descriptions. These row and column descriptions are more generally referred to as line descriptions. Each *line description* describes the number, size and order of the groups of consecutive black cells in the line being described.

A line description d is an ordered list of positive integers $(d_1, d_2, \ldots, d_k)$. Each integer d_i is the number of consecutive black cells of a single group. These groups must be separated by at least one white cell. The first group may be preceded by zero or more white cells and likewise the last group may be followed by zero or more white cells. Each black cell in a line must be part of one of these groups, so the total number of black cells in the line must equal $\sum_{i=1}^{k} d_i$. A line description d can describe a line which can also be described with the regular expression $0^* 1^{d_1} 0^+ 1^{d_2} 0^+ \ldots 0^+ 1^{d_k} 0^*$. Lines without any black cells may be indicated with the special description (0).

A Nonogram N can then be defined as an ordered series of row descriptions $(r_1, r_2, \ldots, r_m)$, describing the rows of I in order from top to bottom and an ordered series of column descriptions $(c_1, c_2, \ldots, c_n)$, describing the columns of I in order from left to right.

Clearly, for each black and white image there is only one Nonogram that properly describes the image. However, a Nonogram may accurately describe multiple black and white images. A requirement for most logic puzzles is that they have precisely one correct solution. As such, a Nonogram is generally only considered a proper Nonogram if it describes precisely one black

and white image. We refer to this subset of Nonograms as *uniquely solvable Nonograms*. As we try to construct a Nonogram from an image, our goal is to construct a uniquely solvable Nonogram.

It is also possibly to construct Nonograms which do not describe any black and white image. Such is the case when the sum of all numbers in the row descriptions is not the same as the sum of all numbers in the column descriptions. We are not interested in such Nonograms as they have no solutions whatsoever. We will not refer to such Nonograms from now on. When we talk about a Nonogram we mean a set of descriptions with one or more solutions.

Given an m by n grayscale image G, we will try to generate one or more uniquely solvable m by n Nonograms. Color images have to be converted to grayscale first and large images should be resized before being turned into Nonograms as large Nonograms are difficult to construct and to solve.

A Nonogram solver is an algorithm that attempts to solve Nonogram puzzles. Given a Nonogram N, it will generate a partial solution. A partial solution P is an m by n matrix contains ones, zeroes and *unknowns*. An *unknown* is a cell whose value a solver has not been able to determine. We will mark unknowns as x throughout this paper. A partial solution P for Nonogram N may only contain ones in cells which hold a one in every valid solution for N and likewise may only contain zeroes in cells which hold a zero in all valid solutions for N. If, for a given field, there exist solutions in which it holds a one and solutions in which it holds a zero, that field must be unknown in P.

Ideally, a solver will generate a partial solution that contains all ones and zeroes that are common to all solutions of N and will only have unknowns in places where some solutions of N have ones whereas other solutions have zeroes. In the case of uniquely solvable Nonograms, that would mean an ideal solver generates the complete solution. However, not all solvers are able to produce such a complete partial solution.

To distinguish Nonograms that a given solver cannot solve from Nonograms that are not uniquely solvable, we introduce solver-specific terminology. If a uniquely solvable Nonogram N can be solved by an algorithm called SOMESOLVER, we say that N is SOMESOLVER-solvable. If a Nonogram is not SOMESOLVER-solvable, it may still be uniquely solvable. In that case, the SOMESOLVER algorithm was apparently not quite thorough enough to solve this specific Nonogram.

In cases where there are multiple solutions for a Nonogram N, we will

refer to the partial solution for N which has the fewest possible number of unknowns as the *optimal partial solution.*

A related problem to Nonograms is the *discrete tomography problem* of constructing a $(0,1)$-matrix from two projections in the form of row and column sums [BK04]. That problem too can produce multiple solutions, more so than Nonograms as there are no restrictions on the number of ones that must appear consecutively in any row or column. In this discrete tomography problem, all solutions can be constructed from any single solution through the application of a limited number of interchanges, simple operations in which two cells holding zeroes and two cells holding ones are exchanged [Rys57]. This basic operation does not change row or column sums and as such, applying an interchange to a solution always produces another solution.

Similar operations can be applied to Nonogram solutions in cases where multiple solutions exist [BK09]. As Nonograms have more restrictions besides the row and column sums having certain values, applying the same simple interchanges to a solution for a Nonogram N may very well create matrices which are not valid solutions for N. Further information about discrete tomography and its applications can be found in [HK99] and [HK07]. The fundamentals of computed tomography in general are covered in [Buz08] and [Her09].

In this paper, we will use the term *switching component.* Given an m by n (partial) solution P, a *switching component* is a set of four cells $P_{ij}, P_{i\ell}, P_{kj}$ and $P_{k\ell}$, where $1 \leq i < k \leq m, 1 \leq j < \ell \leq n$ and $P_{ij} = P_{k\ell} \neq P_{i\ell} = P_{kj}$. In other words, a switching component is a set of four cells in P which form a rectangle with ones on two diagonally opposite corners and zeroes on the other two corners. When a switching component is applied, the values of the four corner cells are swapped, as shown in Figure 2.1.
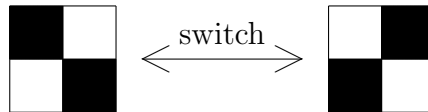

Figure 2.1: A simple switch

Given the optimal partial solution P for Nonogram N, the *flexible set* is the set of all unknowns. For each cell in the flexible set, called *flexible*

cells, there exists at least one solution in which it has value 0 and at least one solution in which it has value 1. A *generalized switching component* is a minimal subset of the *flexible set* for which the values of all cells are directly related to each other.

Generalized switching components of different sizes and shapes have a different number of possible fillings which are allowed by N. We will look into various switching components and how they can be described as ordered combinations of basic switching components. When multiple simple switches are required to transform one solution for N into another, we will consider how the cells involved in those switches can be described as a generalized switching component.

We will also investigate how easily one partial solution can be transformed into another. Given two images I_1 and I_2 which are accurately described by Nonogram N, DISTANCE (I_1, I_2) is the minimal number of switching components that need to be applied to I_1 to create I_2. The distance between two images is only defined for pairs of images that share the same Nonogram description.

During this transformation, the intermediate images may no longer conform to the description N. We can also restrict the intermediate images to those that are described by N. The NONOGRAMDISTANCE (I_1, I_2) is the minimal number of switching components that need to be applied to I_1 to create I_2 such that each intermediate image is valid for N. If no such transformation exists, NONOGRAMDISTANCE $(I_1, I_2) = \infty$. Clearly DISTANCE (I_1, I_2) can never be bigger than NONOGRAMDISTANCE (I_1, I_2), but the former may very well be smaller.

Finally, the *switching graph* of a Nonogram N is a graph in which each solution of N is represented as a vertex and each pair of vertices is connected by an edge if and only if the two Nonogram solutions involved have a DISTANCE of 1.

Chapter 3

Constructing Nonograms from Images

In this chapter, we describe the method we use to construct Nonograms from grayscale images. This method, which is based on the method used in [Kos08], was previously discussed in [BHKP09].

3.1 Method

Using the definitions from the previous chapter, we can describe the general method we use to construct Nonograms based on grayscale images. As can be seen in Algorithm 1, we initialize a candidate puzzle using the grayscale image and then repeatedly adapt that puzzle until it is solvable by a chosen solving algorithm. Any solver can be used on the condition that it does not produce solutions or partial solutions with fixed values chosen for flexible cells. We have chosen to use the solvers described in [BK09].

The first step in constructing a uniquely solvable Nonogram is constructing a candidate puzzle. The function NONOGRAM (I) generates the Nonogram that describes image I. The initialization step focuses mainly on creating a black and white image that looks as much like the grayscale image G as possible. Once this is done, we repeatedly adapt the puzzle one cell at a time until the candidate puzzle is solvable by the solving algorithm we are using. The strength of the solving algorithm can strongly influence the difficulty of the resulting Nonograms. In adapting the puzzle, we need to strike a balance between resembling G and making the candidate solvable more quickly.

ALGORITHM 1: General method for Nonogram construction from a grayscale image G.

---

function CREATEPUZZLE(G)
    $I \leftarrow$ INITIALIZEPUZZLE (G)
    $N \leftarrow$ NONOGRAM (I)
    while not ISSOLVABLE (N) **do**
        $I \leftarrow$ ADAPTPUZZLE (I, G)
        $N \leftarrow$ NONOGRAM (I)
    od
    return N

3.2 Initialize

Grayscale images are generally stored as intensity maps, meaning a value of 0 indicates a black pixel and a value of 2^{n-1} indicates a white pixel, where n is the number of bits per pixel. Values in between are used for various shades of gray. As we want our puzzle to resemble the input image, we want cells in our puzzle to have a value of 1 if the corresponding pixel in the input image has a very low intensity value and a value of 0 if the corresponding pixel in the input image has a very high intensity value.

To achieve this, we apply a threshold to the input image as described in Algorithm 2. Given a threshold t, each cell I_{ij} in our puzzle will be white if the corresponding pixel G_{ij} in the input image has an intensity greater than or equal to t and black otherwise. The threshold can be chosen in several ways. A fixed threshold can be applied to each image, but it is unlikely that a single threshold would work well for many different input images.

A low threshold can be applied and then raised as needed until a solvable puzzle is generated. In some cases this method, shown in Algorithm 3, will yield uniquely solvable puzzles that resemble the input image, but in others no suitable puzzles will be encountered at threshold levels that allow the puzzle to resemble the input image.

This method is guaranteed to produce a uniquely solvable puzzle at some point. When the threshold is raised beyond the intensity range of the input image, the entire puzzle will be black. An entirely black puzzle is a valid Nonogram, though it is trivial to solve and would clearly not resemble most

input images. Using this method, one should also take care not to start with too low a threshold as this leads to puzzles made up of mostly white cells. These puzzles are often trivial to solve, causing the algorithm to stop, while the puzzles may not resemble the input image yet.

ALGORITHM 2: Generate a black and white image by applying a given threshold to a grayscale image.

function APPLYTHRESHOLD(G, t)
    for all i, j **do**
        if $G_{ij} \leq t$ **then**
          $I_{ij} \leftarrow 1$
        else
          $I_{ij} \leftarrow 0$
        fi
    od
    return I

ALGORITHM 3: Construct a uniquely solvable puzzle in the initialization step.

function INITIALIZEPUZZLE(G)
    $t \leftarrow startingThreshold$
    $I \leftarrow$ APPLYTHRESHOLD (G, t)
    $N \leftarrow$ NONOGRAM (I)
    $P \leftarrow$ SOLVE (N)
    while COUNTUNKOWNS $(P) > 0$ **do**
        $t \leftarrow t + 1$
        $I \leftarrow$ APPLYTHRESHOLD (G, t)
        $N \leftarrow$ NONOGRAM (I)
        $P \leftarrow$ SOLVE (N)
    od
    return I

Algorithm 4 is a simple method that works well with the algorithm described in Section 3.1. It aims to fill a certain percentage of the cells in our puzzle. Again, we start with a low threshold and increase that threshold until the resulting puzzle satisfies a given condition.

ALGORITHM 4: Initialize a candidate puzzle with a specified amount of black cells.

function INITIALIZEPUZZLE(G)
    $t \leftarrow 0$
    $I \leftarrow$ APPLYTHRESHOLD (G, t)
    while PERCENTAGEBLACK $(I) < fillTarget$ **do**
        $t \leftarrow t + 1$
        $I \leftarrow$ APPLYTHRESHOLD (G, t)
    od
    return I

It should be noted that unlike the previous method, this method does not necessarily generate uniquely solvable puzzles. It generates candidate puzzles that form the starting point of the method described in Section 3.1. When applied to most grayscale images, the resulting puzzle candidate will have only slightly more than FILLTARGET black cells. However, when applied to black and white images, the result may still be an entirely black puzzle. The advantage of this method is that it is not concerned with generating solvable puzzles, but only with generating a black and white image that resembles the input. The value used for FILLTARGET gives us some control over both the way the puzzle looks and to some degree the expected difficulty of the resulting puzzle.

Once an initial image has been created, some filters may be applied to it. For example, edge detection algorithms may be used, or random noise added, to generate very different puzzles which may still resemble the grayscale image.

3.3 Adapt

After initializing a Nonogram, we need to repeatedly modify it until it is uniquely solvable. This is done by turning one white cell black each time using Algorithm 5, until the puzzle becomes uniquely solvable.

ALGORITHM 5: Generate a black and white image by applying a given threshold to a grayscale image.

---

function ADAPTPUZZLE(I, G)
    $min \leftarrow \infty$
    $N \leftarrow$ NONOGRAM(I)
    $P \leftarrow$ SOLVE(N)
    for all i, j **do**
           if $P_{ij} = x \ \wedge I_{ij} = 0$ **then**
               $I_{ij} \leftarrow 1$
               $N \leftarrow$ NONOGRAM(I)
               $P \leftarrow$ SOLVE(N)
               $value \leftarrow$ EVALUATE(I, G, P)
               if $value < min$ **then**
                   $min \leftarrow value$
                   $k \leftarrow i$
                   $\ell \leftarrow j$
               fi
               $I_{ij} \leftarrow 0$
           fi
    od
    $I_{k\ell} \leftarrow 1$
    return I

The way we decide which cell is colored black each time has a large impact on the algorithm's performance, the difficulty of the resulting puzzles and the similarity between the puzzles and the original images. The EVALUATE function should weigh all the factors we wish to optimize. The closer a puzzle is to being solvable, the lower its return value should be. The larger difference between the input image and the current puzzle, the higher the return value should be. Other factors may be included, which will be discussed later in this section and in subsequent sections.

We know that the puzzle will become solvable at some point if we just keep adding black cells to it, but there is no guarantee that the solving algorithm gets closer to solving the puzzle with each added cell. Each time our solver attempts to solve the puzzle, it will come up with some number of *unknowns*. We can try to maximize the information gained by the solver

with each new black cell by selecting cells that directly affect rows or columns that the solver cannot solve yet. An easy way to achieve this is to limit the ADAPTPUZZLE function to unknowns.

Even the simplest of solvers should have no trouble determining the color of a cell if that cell is the only unknown in a row or column. One can simply count the number of black cells in the line and compare it to the sum of the numbers in the line description. If those numbers are the same, the unknown must be white. If the sum of the line description is one higher than the number of black cells in the line, the cell must be black. As such, an unknown must always be accompanied by another unknown in the same row and one in the same column. This means that by changing an unknown cell from white to black, we directly influence the amount of information that is available regarding at least four cells that are currently unknown to the solver.

There is still no guarantee that the solver will be able to get closer to solving the puzzle after switching a white unknown to black, but we are more likely to this way than by picking just any white cell. Additionally, we can now significantly reduce the number of cells we need to consider as a candidate for switching during each iteration of ADAPTPUZZLE.

3.4   Vary

After constructing a uniquely solvable Nonogram, we can keep adding more black cells to it to construct additional Nonograms that may be easier or more difficult to solve. Algorithm 6 generates a set M of up to *depth* additional puzzles by adding *depth* black cells and checking whether the result of each addition is a solvable Nonogram.

As the method starts with a Nonogram that is solvable, there are no more unknowns and we need to evaluate all white cells. It is possible to create unsolvable Nonograms by adding black cells, but in practice nearly every iteration produces another uniquely solvable Nonogram. This method uses a scoring method similar to that of ADAPTPUZZLE to create puzzles that resemble the original image.

ALGORITHM 6: Generate variations on a Nonogram by adding more black cells.

---

function VARY$(I, G, depth)$
    $M \leftarrow \emptyset$
    $d \leftarrow 0$
    while $d < depth \ \wedge I$ has white cells **do**
        $min \leftarrow \infty$
        $N \leftarrow$ NONOGRAM(I)
        $P \leftarrow$ SOLVE(N)
        for all i, j **do**
            if $I_{ij} = 0$ **then**
                $I_{ij} \leftarrow 1$
                $N \leftarrow$ NONOGRAM(I)
                $P \leftarrow$ SOLVE(N)
                $value \leftarrow$ EVALUATE(I, P, G)
                if $value < min$ **then**
                    $min \leftarrow value$
                    $k \leftarrow i$
                    $\ell \leftarrow j$
                fi
                $I_{ij} \leftarrow 0$
            fi
        od
        $I_{k\ell} \leftarrow 1$
        $N \leftarrow$ NONOGRAM(I)
        if ISSOLVABLE(N) **then**
            $M \leftarrow M \cup \{I\}$
            $d \leftarrow d + 1$
        fi
    od
    return M

3.5   Other options

As mentioned in Section 3.3, only evaluating unknowns, cells for which the solver cannot deduce the correct value, can significantly improve the algorithm's efficiency. Not only are there fewer cases to process, leading to fewer calls of the solving algorithm, but by always switching unknowns, we add more information to the puzzle in each iteration than if we were to switch cells in areas of the puzzle which the solver can already easily solve.

However, there are downsides to limiting ourselves to unknowns. If most of the unknowns in a candidate puzzle correspond to very bright pixels in the input image, we will likely be adding black spots to our final puzzle in places where the original image had none. This may not always be a problem. A few stray cells may allow us to construct a uniquely solvable puzzle that more closely resembles the input image. However, in other cases we may end up adding a lot of black cells that make the input image almost unrecognizable. This is always a possibility when the EVALUATE function tries to not only optimize the similarity between the puzzle and the input image, but also the solvability of the puzzle. But when we strictly limit ourselves to unknowns, we may be ruling out cells which could potentially bring the candidate puzzle closer to being solvable without adding stray cells.

As such, we may want to include known cells in our evaluation. We cannot do anything about the added cost of evaluating more candidate cells during each iteration of the ADAPTPUZZLE function. The fact that switching unknowns adds more useful information needed to solve the puzzle can be compensated for by heavily weighing the number of unknowns we would have after switching a candidate cell, or even increasing the fitness value if the candidate cell is unknown.

If we choose to limit ourselves to unknown cells because doing so leads to shorter runtimes, we may run into a particularly clear case of the problem described above. If the initial candidate puzzle contains empty rows or columns, those rows or columns will always remain blank as even the most trivial solving algorithm should be able to deduce the cell values in those lines. This may lead to puzzles in which lines or shapes are clearly split into two by a blank row or column. If this is considered undesirable, we may add one black cell to each empty row or column immediately after the puzzle is initialized. By adding just one cell to an otherwise empty line, it becomes possible for other cells in the same line to be, or later on become, unknown and thus be considered in one or more iterations of VARY. This is a rather

arbitrary decision, but in some cases it may allow the creation of puzzles that more closely resemble the input image. As the initialization of the puzzle is based solely on the gray values of the input image, it would make sense to avoid empty lines by selecting the darkest pixels in these lines and making them black in our candidate puzzle.

In addition to using VARY to generate multiple puzzles, we may want to repeat the entire algorithm several times. By keeping track of all previously generated puzzles we can avoid generating identical puzzles in each generation. This can be done by assigning a negative weight to cells in the EVALUATE function based on the frequency with which that cell has been black in previously generated puzzles.

3.6 Parameters

We will now briefly discuss all the parameters available in our implementation of the method described in previous sections. Some parameters will have arguments listed between square brackets. If a parameter has a default value which is used if that parameter is not explicitly included, that default value is listed between parentheses at the end of the parameter description. Our program uses the solvers described in [BK09]. Several parameters listed here are used by those solvers, or used to determine which of those solvers to use. The available parameters are:

minthresh [integer] The minimum threshold applied to the grayscale image to produce a black and white image. (default: 0)

maxthresh [integer] The maximum threshold applied to the grayscale image to produce a black and white image. (default: 255)

stepthresh [integer] The step size used to increment the threshold. (default: 1)

level [integer] The level parameter used by the solver. (default: 0)

nopng Do not produce png images. By default, each uniquely solvable Nonogram created by the program is saved as a png image.

nodescr Do not produce plain text Nonogram description files. By default, each uniquely solvable Nonogram created by the program is saved as a text file containing row and column descriptions.

verbose Produce extra output, such as the evaluation results of cells in AdaptPuzzle.

keep_going [integer] After producing a uniquely solvable Nonogram, keep applying AdaptPuzzle to the image until the specified number of uniquely solvable Nonograms has been constructed, or no more Nonograms can be constructed. (default: 0)

analyze Produce additional statistics regarding generated puzzles.

alpha [integer] The weight of unknowns in the cell evaluation function. (default: 8)

beta [integer] The weight of gray value difference in the cell evaluation function. (default: 1)

gamma [integer] The weight of similarity to previous puzzles in the cell evaluation function. (default: 1)

repeat [integer] Number of times the entire algorithm, including the *keep_going* loop, should be repeated after the initial run. (default: 0)

seed [integer] Optional random seed. (default: current POSIX time)

choose [method][method_arguments] Method used to initialize the puzzle, with relevant arguments (default: *solvable*)

adapt [method][method_arguments] Method used to add cells to the puzzle, with relevant arguments (default: *basic*)

filter [method][method_arguments] Filter applied to the puzzle after initialization, with relevant arguments (default: *none*)

fill_lines [method] Method used to add black cells to empty lines. (default: *none*)

Several methods are available for the last four parameters, some having further arguments to determine the precise functionality of those methods. We will now list each of those methods including any arguments.

The different methods available for *choose* are:

choose solvable INITIALIZEPUZZLE raises the threshold until the resulting puzzle is uniquely solvable.

choose fill [integer] INITIALIZEPUZZLE raises the threshold until the puzzle has at least the specified percentage of black cells.

The different methods available for *adapt* are:

adapt basic ADAPTPUZZLE only considers the gray values of individual pixels when choosing a cell in ADAPTPUZZLE.

adapt area [integer] ADAPTPUZZLE considers the gray values of pixels within the specified radius when choosing a cell in ADAPTPUZZLE.

The different methods available for *filter* are:

filter none No filter is applied.

filter contour At the end of INITIALIZEPUZZLE, only keep the contours of black shapes in the black and white image.

filter checker At the end of INITIALIZEPUZZLE, apply a checkered pattern to black shapes in the black and white image.

filter random [integer] At the end of INITIALIZEPUZZLE, randomly keep the specified percentage of black cells in black shapes in the black and white image. Other cells are made white.

The different methods available for *fill_lines* are:

fill_lines none Empty lines are kept empty.

fill_lines once Select one cell in each empty line and color it black. This cell will be black throughout the entire run of the program, including any *repeat* loops.

fill_lines adapt Use ADAPTPUZZLE to add a black cell to each empty line. A different cell may be selected during the initialization step if *repeat* is used.

3.7 Results

Given the algorithm described in this chapter, with the parameter options listed in the previous section, we will perform several runs with different parameters. We will include some of the Nonograms constructed this way along with their respective difficulties. In these tests, we will use the image of Alan Turing in Figure 3.1 as our base image. That image is 55 pixels wide and 70 pixels high. As these dimensions are rather large for a Nonogram, we first scale the image to 20 by 25 pixels, as shown in Figure 3.2.

Figure 3.1: Original image of Alan Turing

Figure 3.2: Original image scaled to 20×25

During the first run, we use the following parameters:

alpha 8		**fill_lines** adapt	
beta 1		**level** 1	
gamma 64		**repeat** 9	
choose *fill* 30		**keep_going** 40	

First, INITIALIZEPUZZLE generates the image in Figure 3.3. The image contains empty lines, which are filled differently for each *repeat*. Figure 3.4 shows the first set of constructed Nonograms. The leftmost image has had a

19

black cell added to each empty line, but is not yet a solvable Nonogram. The second image is the first solvable Nonogram that has been generated. The next four images are created by adding more pixels to the solvable Nonogram using ADAPTPUZZLE until 40 more puzzles have been constructed. The four puzzles shown are the 10th, 20th, 30th and 40th resulting Nonograms.

Figure 3.5 shows the second set of constructed Nonograms. Once again, the leftmost image is the result of adding a black cell to each empty line. As we use ADAPTPUZZLE to select these cells, the resulting image is different from that in Figure 3.4. This Nonogram is also not solvable by our solver with the provided parameters, so more cells are colored black until we find a solvable Nonogram, which is the second image in Figure 3.5. The four images to the right of it are the 10th, 20th, 30th and 40th solvable Nonograms constructed by adding more pixels to the already solvable Nonogram. Towards the end, the image becomes difficult to recognize because the algorithm tries to construct Nonograms that do not look like the Nonograms in Figure 3.4.

Figure 3.6 shows the initial solvable Nonograms constructed in six further repeats and Figure 3.7 shows the final solvable Nonograms constructed in each of those repeats through the use of *keep_going*.
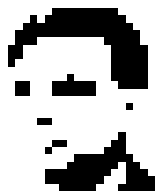

Figure 3.3: Result of INITIALIZEPUZZLE
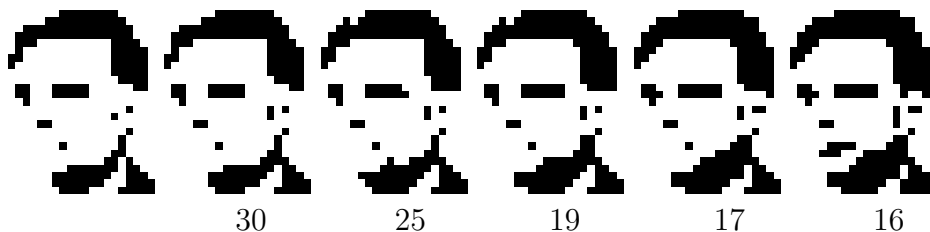
Figure 3.4: Initial Nonograms, from left to right: lines filled, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times

26   15   22   32   18
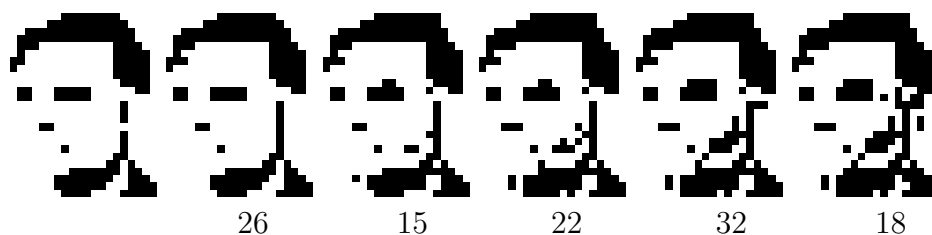
Figure 3.5: First repeat, from left to right: lines filled, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times

27   30   36   32   30   35
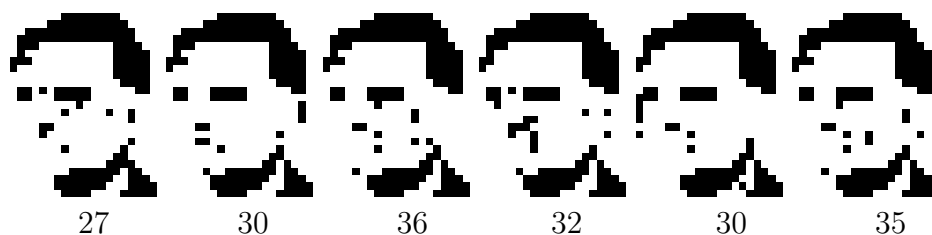
Figure 3.6: From left to right: six more repeats, initial solvable Nonogram

33   18   21   23   32   31
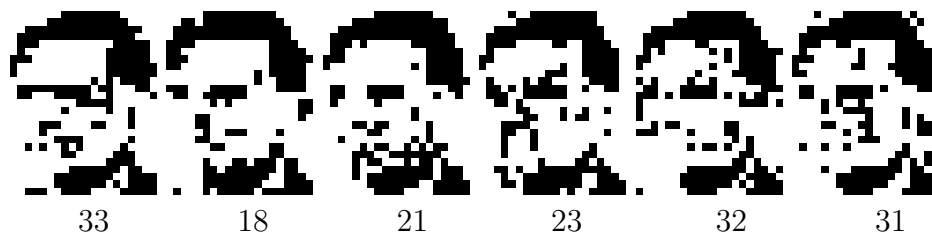
Figure 3.7: From left to right: six more repeats, *keep_going* 40 times

During the second run, we use the following parameters:

alpha 8		**fill_lines** adapt	
beta 1		**level** 1	
gamma 64		**repeat** 9	
choose *fill* 35		**keep_going** 40	

Figure 3.8 shows the result of INITIALIZEPUZZLE. Compared to Figure 3.3, this image has more black cells, because of the higher *fill* parameter. Once again, this image contains empty lines which are filled differently for each *repeat.* Figure 3.9 shows the first set of constructed Nonograms. The leftmost image has had a black cell added to each empty line and once more the resulting Nonogram is not solvable by our chosen solver. The second image is the first solvable Nonogram generated by adding black cells. The four image to the right of that are 10th, 20th, 30th and 40th Nonograms constructed through the use of *keep_going.*

Figure 3.10 shows the second set of constructed Nonograms. Once again, the leftmost image is the result of adding a black cell to each empty line. The resulting image is the same as in Figure 3.9 despite our use of ADAPTPUZZLE to select these cells. Apparently the penalty for resembling a previously constructed Nonogram does not outweigh other factors, such as similarity to the input image and the number of unknowns in the resulting Nonogram.

The resulting Nonogram is also not solvable by out solver with the provided parameters, so more cells are colored black until we find a solvable Nonogram, which is the second image in Figure 3.10. The four images to the right of it are the 10th, 20th, 30th and 40th solvable Nonograms constructed by adding more pixels to the already solvable Nonogram.

Figure 3.11 shows the initial solvable Nonograms constructed in six further repeats and Figure 3.12 shows the final solvable Nonograms constructed in each of those repeats through the use of *keep_going.* Once again, later *repeats* tend to construct Nonograms which do not resemble the input image as much as Nonograms constructed earlier.
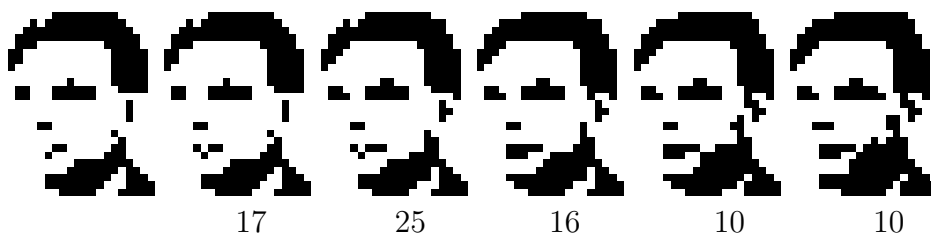
Figure 3.8: Result of INITIALIZEPUZZLE

| 17 | 25 | 16 | 10 | 10 |

Figure 3.9: Initial Nonograms, from left to right: lines filled, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times
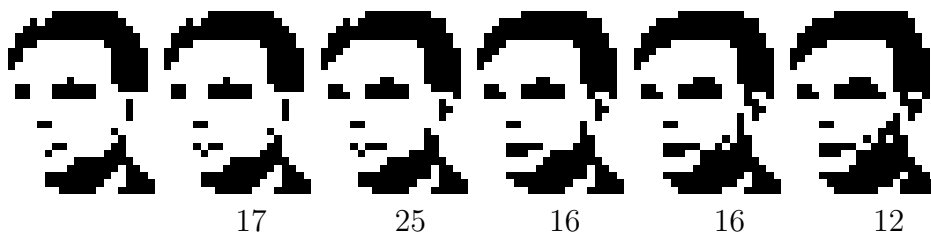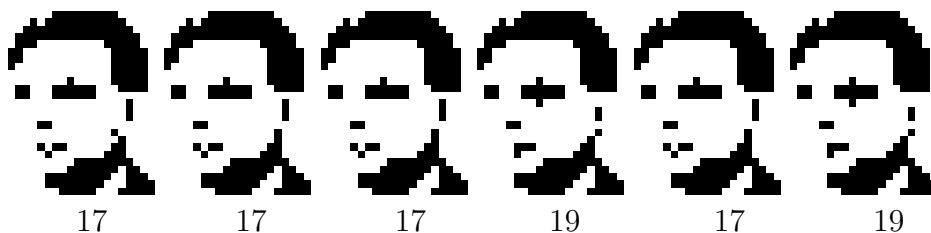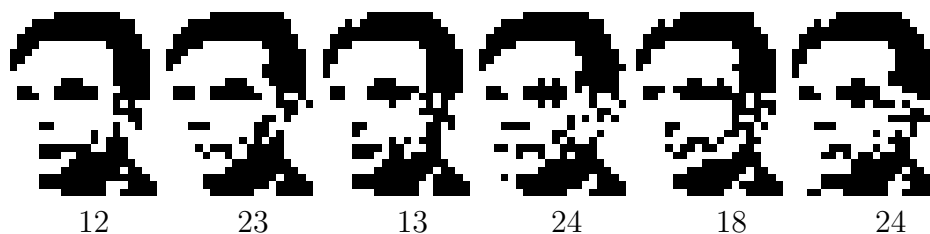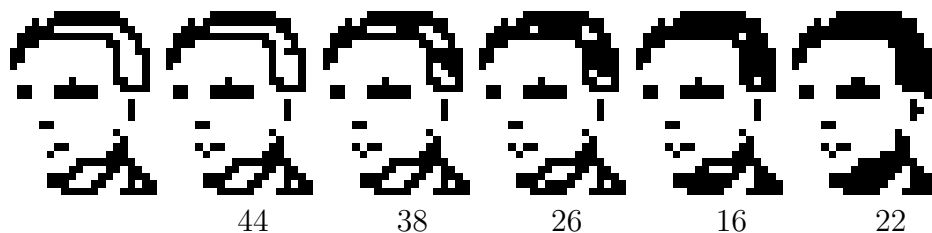

| 17 | 25 | 16 | 16 | 12 |

Figure 3.10: First repeat, from left to right: lines filled, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times

| 17 | 17 | 17 | 19 | 17 | 19 |

Figure 3.11: From left to right: six more repeats, initial solvable Nonogram

| 12 | 23 | 13 | 24 | 18 | 24 |

Figure 3.12: From left to right: six more repeats, *keep_going* 40 times

During the third run, we use the following parameters:

alpha 8	**fill_lines** adapt
beta 1	
gamma 64	**level** 1
choose *fill* 35	**repeat** 9
filter contour	**keep_going** 40

During our third run, we use a filter. Other than that, all parameters are the same as during the previous run. Figure 3.13 shows the result of INITIALIZEPUZZLE, which is identical to Figure 3.8. After a black cell has been added to each empty line in this image, we apply a very simple filter which colors all cells which are surrounded by black cells white. As we once again the black cells added to empty lines are chosen separately for each *repeat*, the filter must also be applied anew during each *repeat*.

Figure 3.14 shows the first set of constructed Nonograms. The leftmost image has had a black cell added to each empty, followed by the filter. The second image is the first solvable Nonogram generated by adding black cells to the first image. Thanks to the filter, this Nonogram is clearly very different from all previously generated Nonograms based on the same grayscale image. The following four images are the 10th, 20th, 30th and 40th Nonograms constructed through the use of *keep_going*. Most black cells added this way are found in the portrait's hair, which was made white by the filter. This is no coincidence, as those cells are all very dark in the grayscale image.

Figure 3.15 shows the second set of constructed Nonograms. The leftmost image is the result of adding a black cell to each empty line and applying a filter. The resulting image is slightly different from that in Figure 3.14. The

second image is the first solvable Nonogram generated by applying ADAPT-PUZZLE. The remaining four images are once again the 10th, 20th, 30th and 40th Nonograms constructed thanks to *keep_going*. These Nonograms are similar to those from the set due to the EVALUATE function's optimization towards Nonograms that resemble the grayscale image, though some small differences occur thanks to the optimization towards Nonograms that do not resemble previously constructed Nonograms.

Figure 3.16 shows the initial solvable Nonograms constructed in six further repeats and Figure 3.17 shows the final solvable Nonograms constructed in each of those repeats through the use of *keep_going*. Later *repeats* have fewer black cells added to the hair part of the picture as a result of the high *gamma* parameter and the final Nonograms constructed are increasingly difficult to recognize as the grayscale image.

It is also interesting to note that the difficulty levels in the third run tend to be higher than those in the first run, which in turn tend to be higher than those in the second run. This is largely due to the number of black cells and the presence of large groups in the images. Nonograms with many black cells tend to be easier than Nonograms with few black cells, especially if those black cells are grouped together into large black areas. Adding more black cells to a Nonogram with *keep_going* also tends to generally lower the difficulty of the Nonogram. By adding a significant amount of black cells this way and repeating the entire procedure several times, we can usually construct puzzles with a fairly large variety in difficulties.
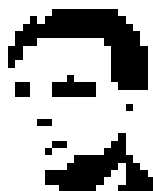
Figure 3.13: Result of INITIALIZEPUZZLE

44   38   26   16   22

Figure 3.14: Initial Nonograms, from left to right: filtered, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times
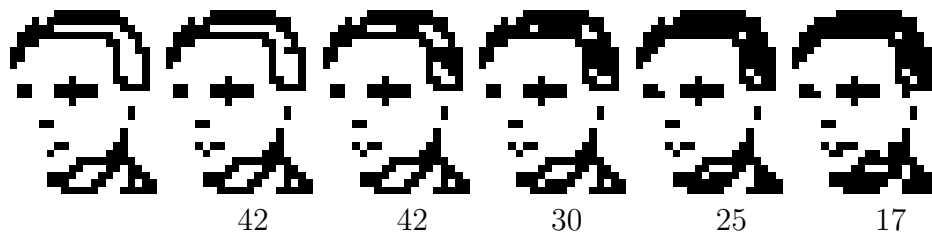

42   42   30   25   17

Figure 3.15: First repeat, from left to right: filtered, first uniquely solvable puzzle, *keep_going* 10, 20, 30, 40 times
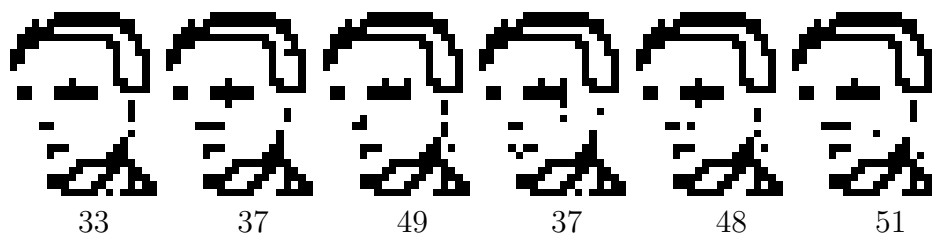

33   37   49   37   48   51
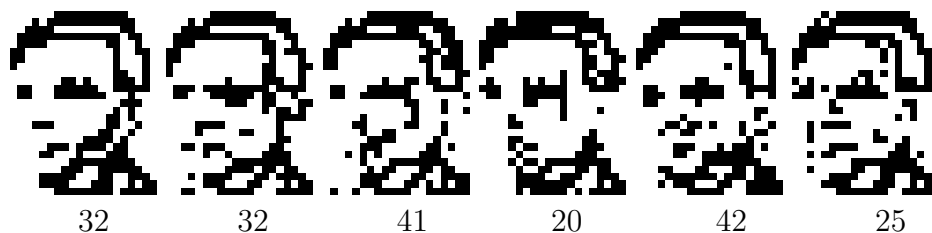
Figure 3.16: From left to right: six more repeats, initial solvable Nonogram

32   32   41   20   42   25

Figure 3.17: From left to right: six more repeats, *keep_going* 40 times

Chapter 4

Switching Components in Nonograms

The discrete tomography problem mentioned in Chapter 2 is very similar to the problem of solving Nonograms. The only difference is that where Nonograms specify groupings of black cells (i.e., cells containing ones) in lines, the discrete tomography problem only specifies the total number of ones in lines. Given an instance of that problem, any two solutions can be transformed into one another through the use of switching components. In fact, by taking any solution and flipping the values of a switching component inside the solution, the result is guaranteed to also be a solution, as switching components do not change the row or column sums [Rys57].

The same does not hold true for Nonograms. Because Nonograms place more restrictions on the values in a matrix, activating a switching component (i.e. changing the zeroes into ones and the ones into zeroes) is likely to result in a matrix that is not described by the same Nonogram as before the switch. In [BK09], the notion of more complex switching components, referred to as generalized switching components, is introduced. Given a Nonogram with multiple solutions, the various solutions may be transformed into one another through the activation of generalized switching components, just as solutions for the discrete tomography problem may be transformed into one another through the application of simple switching components. It is important to note that, unlike in the discrete tomography problem, the presence of generalized switching components in Nonograms does not just depend on the values of the flexible cells themselves, but also on the values of the cells in between or directly adjacent to flexible cells.

In an attempt to find and define different kinds of generalized switching components, we will generate all solutions for some Nonograms that have multiple solutions. We will then determine the flexible set, determine the distance and NonogramDistance between pairs of solutions.

4.1   Calculating distance

We compute the distance between Nonogram solutions with a simple recursive algorithm. Given puzzles p_1 and p_2, we find all switching components in p_1. For each switching component, we compute p_1' in which the values of the switching component have been switched. We then recurse and compute the distance between p_1' and p_2. If at any point, the two puzzles in question are identical, the distance between them is 0.

By keeping track of the shortest transformation found so far, we can recurse whenever the number of activated switches in the current transformation exceeds that in the shortest. We also keep track of all switches that have been performed and disallow them from being repeated to avoid loops in which a single switching component is repeatedly applied.

Most of the arguments are only used in recursive calls. In the initial call, the only required arguments are the two solutions p_1 and p_2. The current recursion depth d is 0 in the initial call and the set S of previously applied switching components is empty. The best result found so far, min, can safely be set to ∞, though if we know some upper bound for the distance between p_1 and p_2, it can be supplied during the initial call to avoid unnecessary work.

Each switch that is performed during the transformation from p_1 to p_2 changes four values from zero to one or from one to zero. If we define the difference between p_1 and p_2 as the number of (i, j) for which $p_{1ij} \neq p_{2ij}$, each switch can change the difference by any even number between -4 and 4. It might seem like only switches which reduce the difference should be performed, however there are cases where p_1 can only be transformed into p_2 by performing one or more switches which do not change, or even increase the difference. One example of such a case can be seen in Figure 4.1.

function APPLYSWITCH(p, i_1, j_1, i_2, j_2)

    $p_{i_1j_1} \leftarrow p_{i_1j_2}$

    $p_{i_1j_2} \leftarrow p_{i_2j_2}$

    $p_{i_2j_1} \leftarrow p_{i_2j_2}$

    $p_{i_2j_2} \leftarrow p_{i_1j_1}$

function DISTANCE$(p, q, d = 0, min = \infty, S = \emptyset)$

    if $d \geq min$ **then**

        return ∞

    else if $p = q$ **then**

        return d

    fi

    for $i_1 \leftarrow 1$ **to** $m - 1$ **step** 1 **do**

        for $j_1 \leftarrow 1$ **to** $n - 1$ **step** 1 **do**

            for $i_2 \leftarrow i_1 + 1$ **to** m **step** 1 **do**

                if $(p_{i_1j_1} = 0 \wedge p_{i_2j_1} = 1) \vee (p_{i_1j_1} = 1 \wedge p_{i_2j_1} = 0)$ **then**

                    for $j_2 \leftarrow j_1 + 1$ **to** n **step** 1 **do**

                        if $p_{i_1j_1} = p_{i_2j_2} \wedge p_{i_1j_2} = p_{i_2j_1} \wedge (i_1, j_1, i_2, j_2) \not\subset S$ **then**

                            APPLYSWITCH(p, i_1, j_1, i_2, j_2)

                            $S \leftarrow S \cup \{(i_1, j_1, i_2, j_2)\}$

                            $d' \leftarrow$ DISTANCE$(p, q, d + 1, min, S)$

                            if $d' < min$ **then**

                                $min \leftarrow d'$

                            fi

                            APPLYSWITCH(p, i_1, j_1, i_2, j_2)

                            $S \leftarrow S \setminus \{(i_1, j_1, i_2, j_2)\}$

                        fi

                    od

                fi

            od

        od

    od

    return min
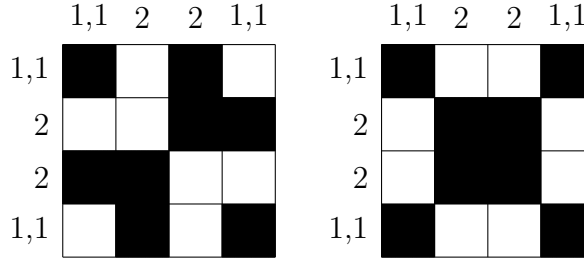
Figure 4.1: Example of two solutions of a Nonogram which require a switch which does not decrease the difference between the solutions to transform one into the other

Once we know the distance using simple switching components between all pairs of solutions for a given Nonogram without restricting ourselves to the Nonogram solution space, we can easily determine the same while being restricted to that space. All direct transformations between solutions through a single simple switching component are reflected in the distance matrix by a distance of 1. By setting all other distances to ∞ we ignore all transformations that violate the Nonogram description. By then applying the Floyd-Warshall algorithm we find the NonogramDistances between all pairs of solutions.

4.2   Generalized switching components

Figure 4.2 shows one possible generalized switching component consisting of six flexible cells a, b, c, d, e and f. This particular type of generalized switching component can be satisfied in two ways: $a = d = f = 1 \land b = c = e = 0$, or $a = d = f = 0 \land b = c = e = 1$.

The cells adjacent to, but not enclosed by flexible cells should be 0. To prove this, assume a case where such a cell is 1. Using Figure 4.2 as an example, we will assume the cell to the immediate left of a is 1. Given that the value of all cells to the left of a are independent of the value of a, we can determine exactly what group the cell to the left of a is part of and how many consecutive cells to the left of a are 1. If that number is the same as the required size of that group, a must be 0. Otherwise, a must be 1. As we cannot determine the value of a, the cell immediately to the left of a cannot be 1.

It is also possible for generalized switching components to appear at the

edge of a Nonogram, in which case there is no cell adjacent to a flexible cell in some direction. As additional zeroes to the left or right of a line segment does not change the description of that line segment, we will sometimes refer to the zero to the left or right of a flexible cell regardless of whether there is actually a zero to the left or right of that cell.

The value of the cells directly between two flexible cells should conform to some pattern that ensures that the line description of the line involved is the same for all solutions of the Nonogram.

Given two flexible cells a and b, themselves not enclosed by flexible cells, which enclose a string S, the (partial) line description of $01S00$ must be identical to that of $00S10$. The only two patterns for which this is always the case are 1^* and $(0^+1)^*0^+$. In the first case, given $S = 1^n$ where $n = |S|$, the description of $01S00 = 0^+1^{n+1}0^+$. The description of $00S10 = 0^+1^{n+1}0^+$. This is the case because in Nonograms, the number and grouping of ones is important, but the number and grouping of zeroes is not. We only require one or more zeroes between groups of ones.

In the second case, given $S = (0^+1)^n0^+$, where $n = \sum_{i=1}^{|S|} S_i$, the description of $01S00 = (0^+1)^{n+1}0^+$. The description of $00S10 = 0^+1^{n+1}0^+$. Once again, the description of $0aSb0$ is identical whether $a = 0 \wedge b = 1$ or $a = 1 \wedge b = 0$.

$$
\begin{array}{ccccccc}
0 & & & & & 0 & \\
0 & a & \cdot & \cdot & \cdot & \cdot & \cdot & b & 0 \\
 & \cdot & & & & & \cdot \\
 & \cdot & & 0 & & & \cdot \\
0 & c & \cdot & \cdot & d & 0 & & \cdot \\
0 & & & \cdot & & & \cdot \\
 & & & \cdot & & & \cdot \\
 & & 0 & e & \cdot & \cdot & f & 0 \\
 & & & 0 & & & 0 \\
\end{array}
$$
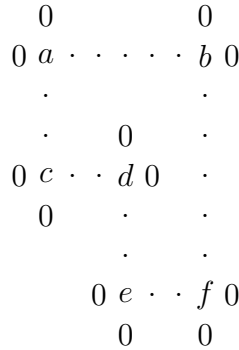
Figure 4.2: Example of a generalized switching component

As we are interested in all solutions for all m by n Nonograms, we can easily generate all m by n images and split them into groups by generating the Nonogram description for each of those images. Each group represents a Nonogram N and the images inside that group are all solutions for N. Groups consisting of one image are uniquely solvable Nonograms. Images that

are placed in the same group can be transformed into one another through switching components and we will look at those groups of images. For each of these groups, we will construct the *switching graph*. We will then count occurrences of different non-isomorph *switching graphs*.

4.3 Results

Figure 4.3 shows the number of 4 by 4 Nonograms with a certain number of flexible cells, as well as the combined number of solutions of all Nonograms with the same number of flexible cells. The number of Nonograms with a certain number of flexible cells and the total number of solutions for these Nonograms are not directly related, as the number of flexible cells itself is not directly related to the number of solutions. The only two exceptions are the first two lines in the table: Nonograms with zero flexible cells have only one solution, thus the number of solutions is equal to the number of Nonograms, and Nonograms with four flexible cells always have exactly two solutions, thus the number of solutions is twice the number of Nonograms. Note that this does not mean there are no Nonograms with two solutions and more than four flexible cells. In fact, such Nonograms are fairly common. One example is the Nonogram with the description $1, 1$ for each line, which has two checkerboard solutions.

flexible cells	number of Nonograms	total number of solutions
0	52 362	52 362
4	4 396	8 792
6	84	168
7	236	708
8	625	1 524
9	48	192
10	68	224
11	104	436
12	180	560
14	64	352
16	29	218
sum	58 196	$2^{16} = 65\,536$

Figure 4.3: Number of 4×4 Nonograms and solutions with given number of unknowns

Figures 4.4 and 4.5 contain the same information as Figure 4.3, but for 5 by 4 Nonograms and 5 by 5 Nonograms respectively.

flexible cells	number of Nonograms	total number of solutions
0	814 632	814 632
4	70 894	141 788
6	3 230	7 082
7	3 532	10 596
8	13 398	34 108
9	708	2 472
10	1 636	5 510
11	1 776	7 320
12	3 314	9 778
13	316	1 480
14	892	4 440
15	344	1 912
16	635	3 474
17	76	484
18	174	1 340
19	40	392
20	146	1 768
sum	915 743	$2^{20} = 1\,048\,576$

Figure 4.4: Number of 5×4 Nonograms and solutions with given number of unknowns

Figures 4.6 and 4.7 show all the different switching graphs for groups of solutions for 4 by 4 Nonograms along with the number of occurrences of each graph. The first entry in the table, type 1, represents uniquely solvable Nonograms. There are 52 362 different uniquely solvable 4 by 4 Nonograms.

Type 2 represents Nonograms with two solutions which cannot be transformed into each other without leaving the Nonogram's solution space, i.e., one solution cannot be transformed into the other through the application of a simple switch. Nonograms of this type include checkerboards and Nonograms whose flexible set consists of a single generalized switching component of the type show in Figure 4.2. As there are 654 4 by 4 Nonograms with this switching graph and each of those Nonograms has two solutions, this entry accounts for 1308 binary 4 by 4 images.

flexible cells	number of Nonograms	total number of solutions
0	25 309 575	25 309 575
4	2 311 785	4 623 570
6	150 638	333 616
7	109 532	328 596
8	485 062	1 255 080
9	24 659	83 088
10	72 592	244 580
11	54 586	221 698
12	125 937	381 790
13	16 748	76 896
14	31 116	144 616
15	20 248	98 148
16	31 898	138 588
17	7 928	50 786
18	9 828	65 608
19	5 890	45 052
20	6 534	58 038
21	2 310	21 856
22	1 970	21 518
23	1 194	16 336
24	708	10 337
25	1 082	25 060
sum	28 781 820	$2^{25} = 33\,554\,432$

Figure 4.5: Number of 5×5 Nonograms and solutions with given number of unknowns
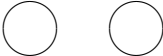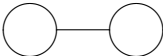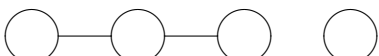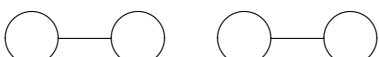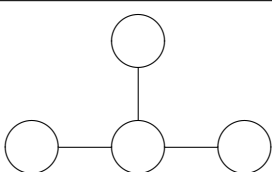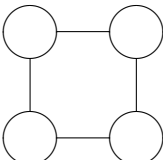
type	switching graph	occurrences
1		52 362
2		654
3		4 396
4		2
5		32
6		388
7		4
8		16
9		72
10		56
11		61

Figure 4.6: Solution switching graphs in 4 by 4 Nonograms
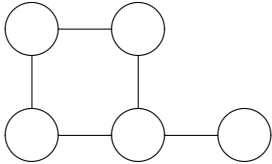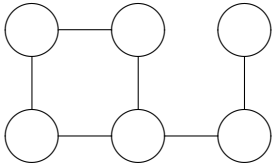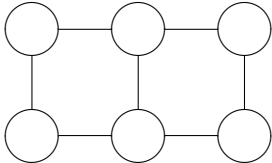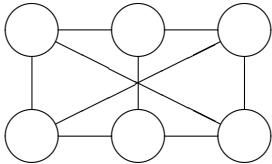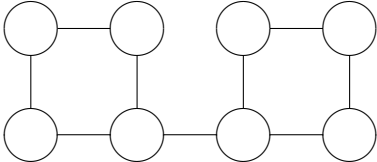
type	switching graph	occurrences
12		20
13		48
14		48
15		16
16		4
17	See Figure 4.8	16
18	See Figure 4.13	1

Figure 4.7: Solution switching graphs in 4 by 4 Nonograms continued
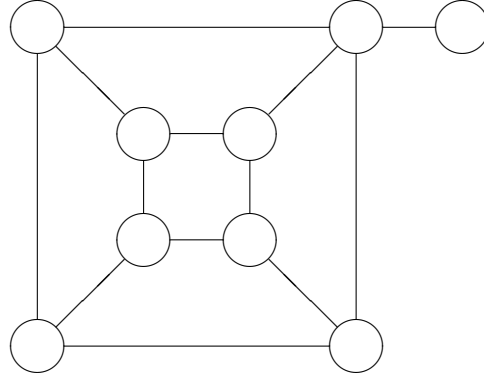
Figure 4.8: Switching graph for a 4 by 4 Nonogram with 9 solutions

Type 3 represents Nonograms with two solutions which can be transformed into each other without leaving the Nonogram's solution space. This means the flexible set of these Nonograms consists of a single simple switching component.

Type 4, consisting of Nonograms with three solutions which each have a distance greater than one to each of the other solutions, only occurs twice. Both Nonograms, with their three difference solutions, are shown in Figures 4.9 and 4.10. In both cases, all cells in the Nonogram are flexible. The first solution of both these Nonograms, as listed below, is the complement of the same Nonogram's third solution. The middle solution of the two Nonograms are each other's inverse.
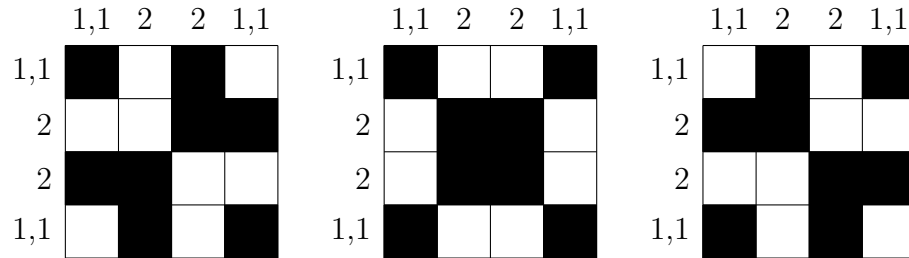
Figure 4.9: A 4×4 Nonogram of type 4

Nonograms of type 12 have 5 solutions. Figure 4.11 shows a Nonogram of this type. The solution on the left in the figure is the solution that can only be transformed into one other solution. This is the only solution in which
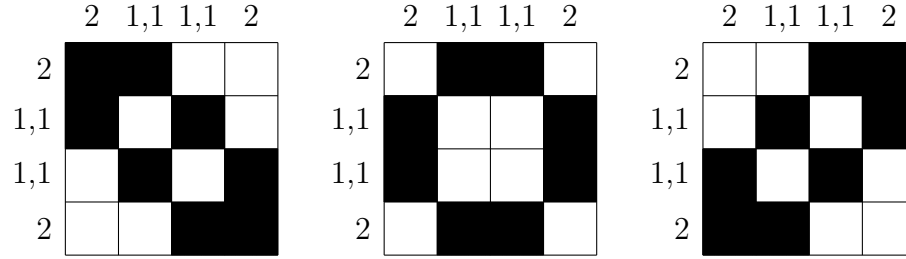
Figure 4.10: A 4×4 Nonogram of type 4

the bottom-right cell is empty. The partial solution on the right in the figure shows what happens when the bottom-right cell is colored black. Cells with an x in partial solutions are cells which can be either black or white, so when the bottom-right cell is colored black in this Nonogram, eight cell values remain unknown.
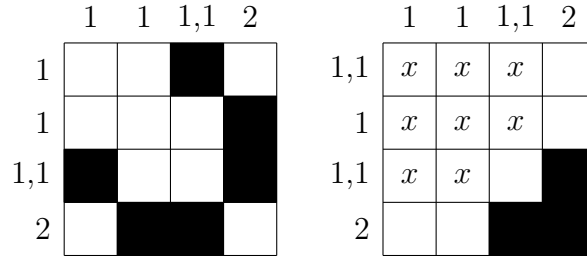
Figure 4.11: A 4×4 Nonogram of type 12

Nonograms of type 17 are similar to those of type 12, in that they both have one solution which can only be transformed into one other solution. The possible transformations between the other eight solutions show a clear structure. Figure 4.12 shows an example of such a Nonogram. The solution which can only be transformed into one other solution is shown on the left in the figure, while the partial solution on the right shows the cells that remain flexible in the remaining eight solutions.

Type 18 consists of one Nonogram, in which each row and each column contains exactly one black cell. The corresponding switching graph is shown in Figure 4.13. This Nonogram is essentially the n-rooks problem on a 4 by 4 board. This leads to a highly regular set of solutions. There are $4! = 24$ solutions. In each solution, any two black cells can be used to perform a
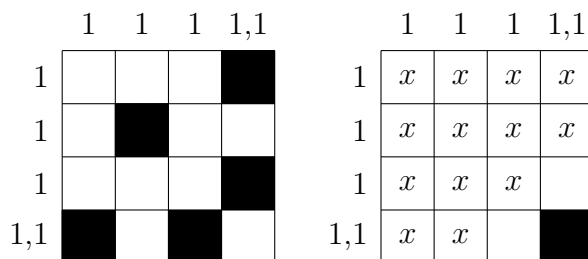
Figure 4.12: A 4×4 Nonogram of type 17

switch, meaning each solution is adjacent two $3 + 2 + 1 = 6$ other solutions. The maximum distance between any two solutions is 3.

The n-rooks problem occurs in all n by n Nonograms. Furthermore, smaller versions of the n-squares problem occur multiple times in Nonograms that are larger than n by n.

The Nonograms of type 15 all consist of instances of the n-rooks problem for $n = 3$. In each case, there are $3! = 6$ solutions, each of which is adjacent to $2 + 1 = 3$ other solutions. The maximum distance between any two solutions is 2. The 3-rooks problem can only occur in a 4 by 4 Nonogram if the unused row and column are both empty. Any of the four rows and any of the four columns can remain unused. As there are 16 row/column combinations, the 3-rooks problem occurs exactly 16 times.

Simple switching components can be seen as instances of the 2-rooks problem. By looking at the many situations in which simple switching components can occur, it becomes clear that the number of occurrences of smaller instances of the n-rooks problem in larger Nonograms cannot be easily described. Rows and columns not involved in the smaller n-rooks problems may still contain some number of black cells, just as long as their value can be determined (i.e., the cells are not flexible) and the black cells do not touch the cells involved in the n-rooks problem.

Figure 4.14 shows some of the switching graphs for groups of solutions for 5 by 4 Nonograms along with the number of occurrences of each of the given graphs. There are 137 different types found in 5 by 4 Nonograms, compared to just 18 in 4 by 4 Nonograms. Note that the type numbering is different between different Nonogram sizes, so a 5 by 4 Nonogram of a given type number will most likely have a different switching graph than a 4 by 4 Nonogram of the same type number.
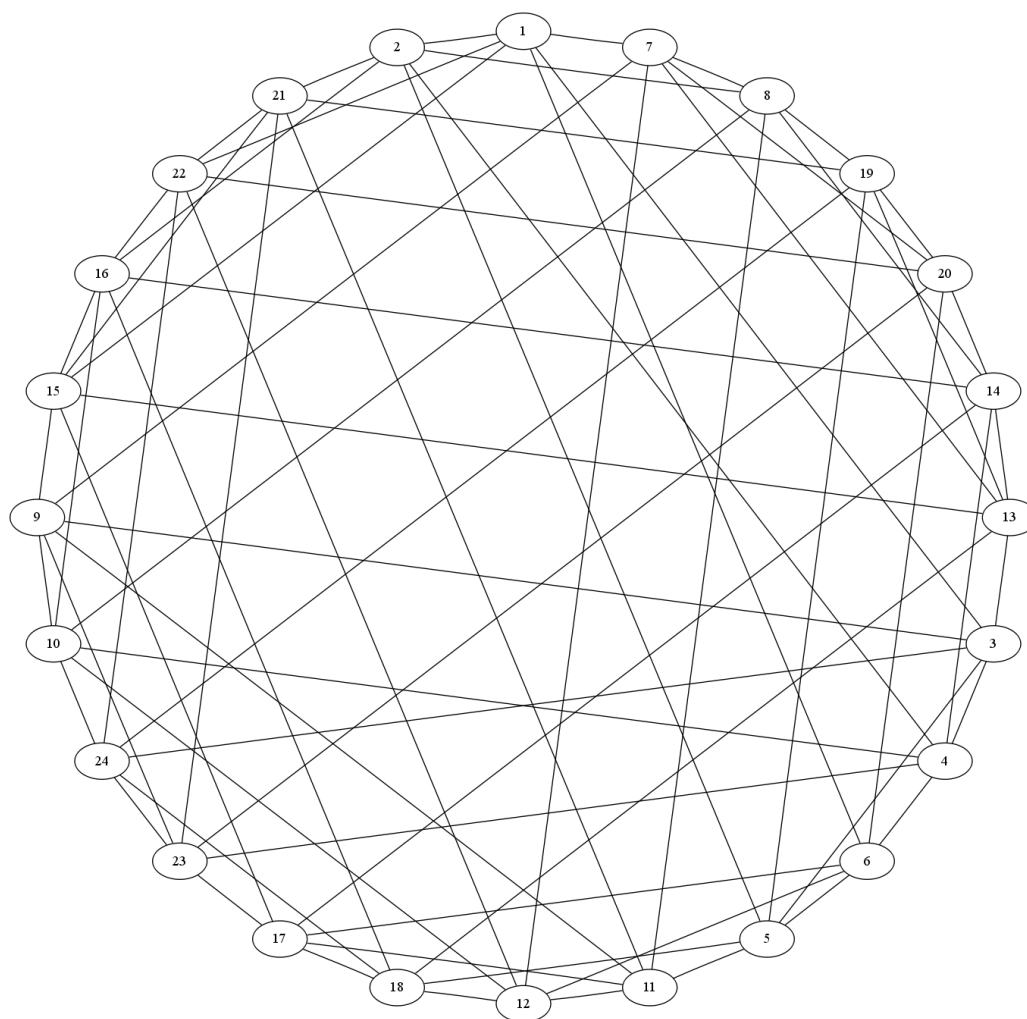
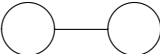Figure 4.13: Switching graph for a 4 by 4 Nonogram with 24 solutions
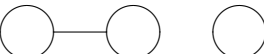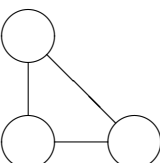
type	switching graph	occurrences
1		814 632
2		13 314
3		70 894
4		160
5		1 500
6		7 328
7		622
8. . . 134	various	7 280
135	see Figure 4.13	5
136	see Figure 4.18	4
137	see Figure 4.19	4

Figure 4.14: Some solution switching graphs in 5 by 4 Nonograms

The first six entries in the table for 5 by 4 Nonograms are the same as for 4 by 4 Nonograms and the relative frequency of each is similar as well.

Nonograms of type 7 have three solutions, each of which can be transformed into both of the other solutions through a single switch. An example of this type of Nonogram, with all three solutions, is shown in Figure 4.15. This type of Nonogram involves flexible cells in three non-adjacent rows or columns and as such can only occur in Nonograms with a width or height greater than four.
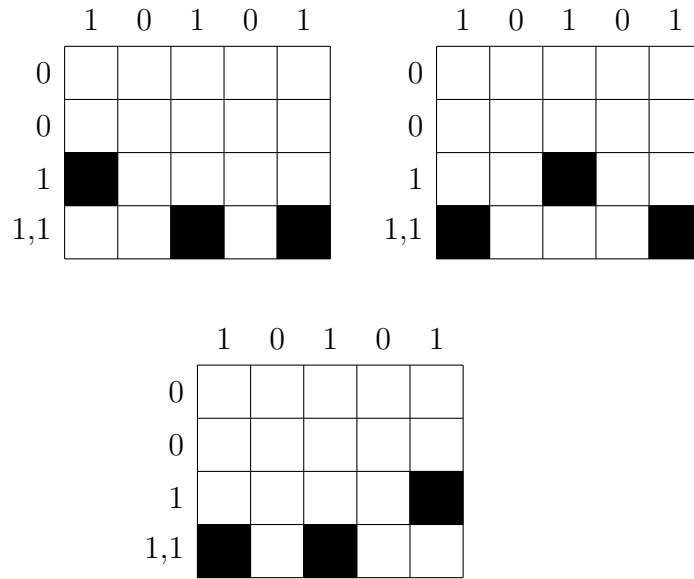
Figure 4.15: A 5×4 Nonogram of type 7

Nonograms of types 8 through 134 are not included. These types account for 7 280 different 5 by 4 Nonograms with a combined total of 36 338 solutions.

The 5 by 4 Nonograms of type 135 are almost the same as the 4 by 4 Nonogram of type 18. They contain the 4-rooks problem, the difference being that there is a spare column which is not used at all. As any one of the five columns can be left empty, leaving the remaining four to form the 4-rooks problem, this type of Nonogram occurs five times.

Nonograms of type 136 are very similar to those of type 135. An example of a Nonogram of this type is shown in Figure 4.16. The difference with Nonograms of type 135 is that instead of keeping one column empty, one pair of adjacent columns must be matched to one specific row. There are four 5

by 4 Nonograms of this type, each with the line description (2) assigned to a different row. However, despite the similarities, the solutions of Nonograms of type 136 cannot be transformed into one another the same way. While each solution of a Nonogram of type 135 can be transformed into six other solutions through single switches, solutions of Nonograms of type 136 can only be transformed into four of five other solutions through switches, as can be seen in figure 4.18.
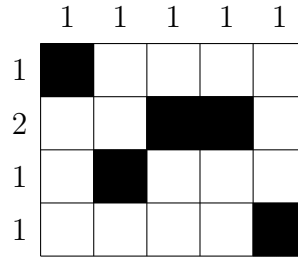
Figure 4.16: One possible solution for a 5×4 Nonogram of type 136

An example of a Nonogram of type 137 can be seen in Figure 4.17. These Nonograms have 36 solutions, the largest number of solutions possible in 5 by 4 Nonograms. This type of Nonogram occurs four times, each time with a different row having the line description $(1, 1)$.
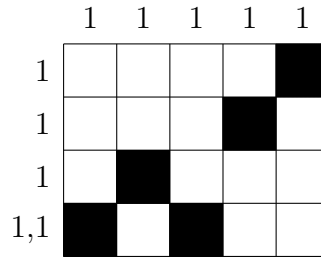
Figure 4.17: One possible solution for a 5×4 Nonogram of type 137

Figure 4.20 shows some of the switching graphs for groups of solutions for 5 by 5 Nonograms along with the number of occurrences of each of the given graphs. There are 1916 different types of 5 by 5 Nonograms when looking only at switching graphs.

The first seven entries in the table for 5 by 5 Nonograms are the same as for 5 by 4 Nonograms and the relative frequency of each is similar as well.
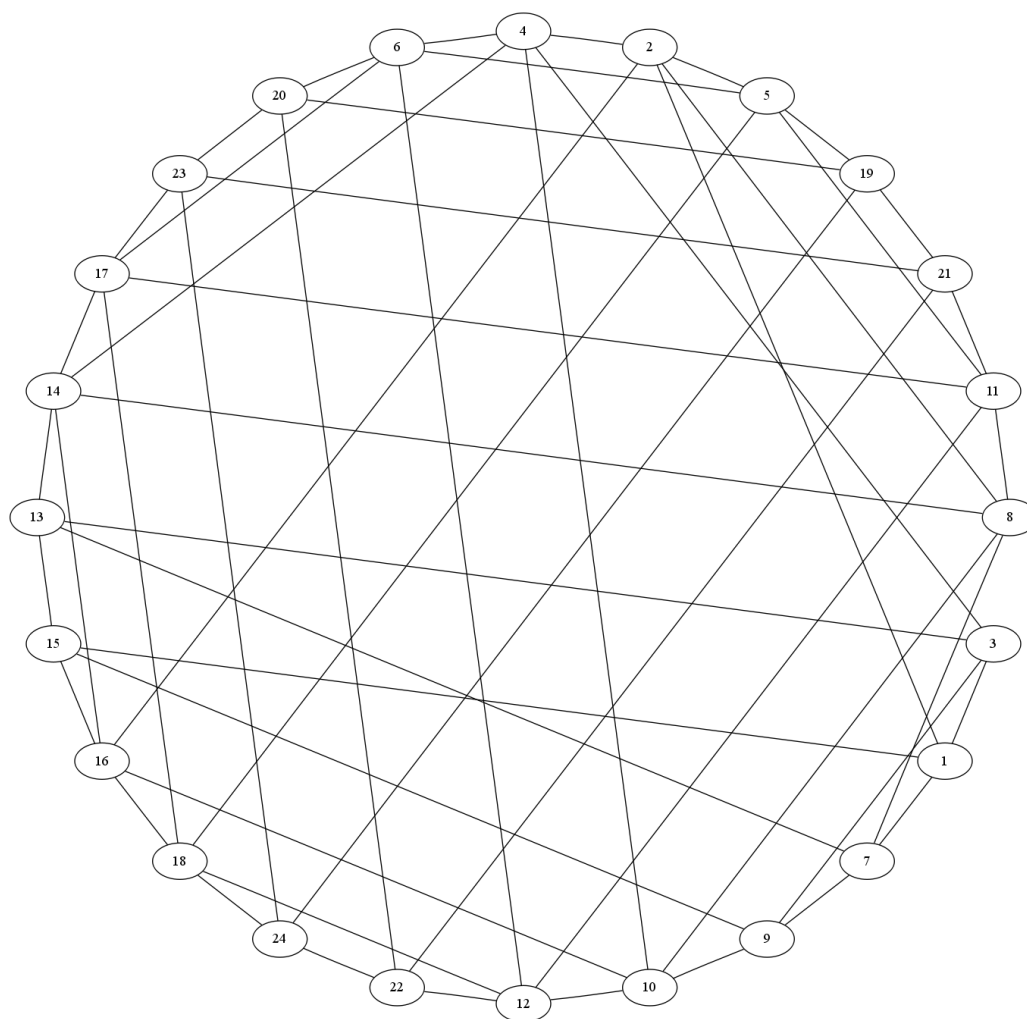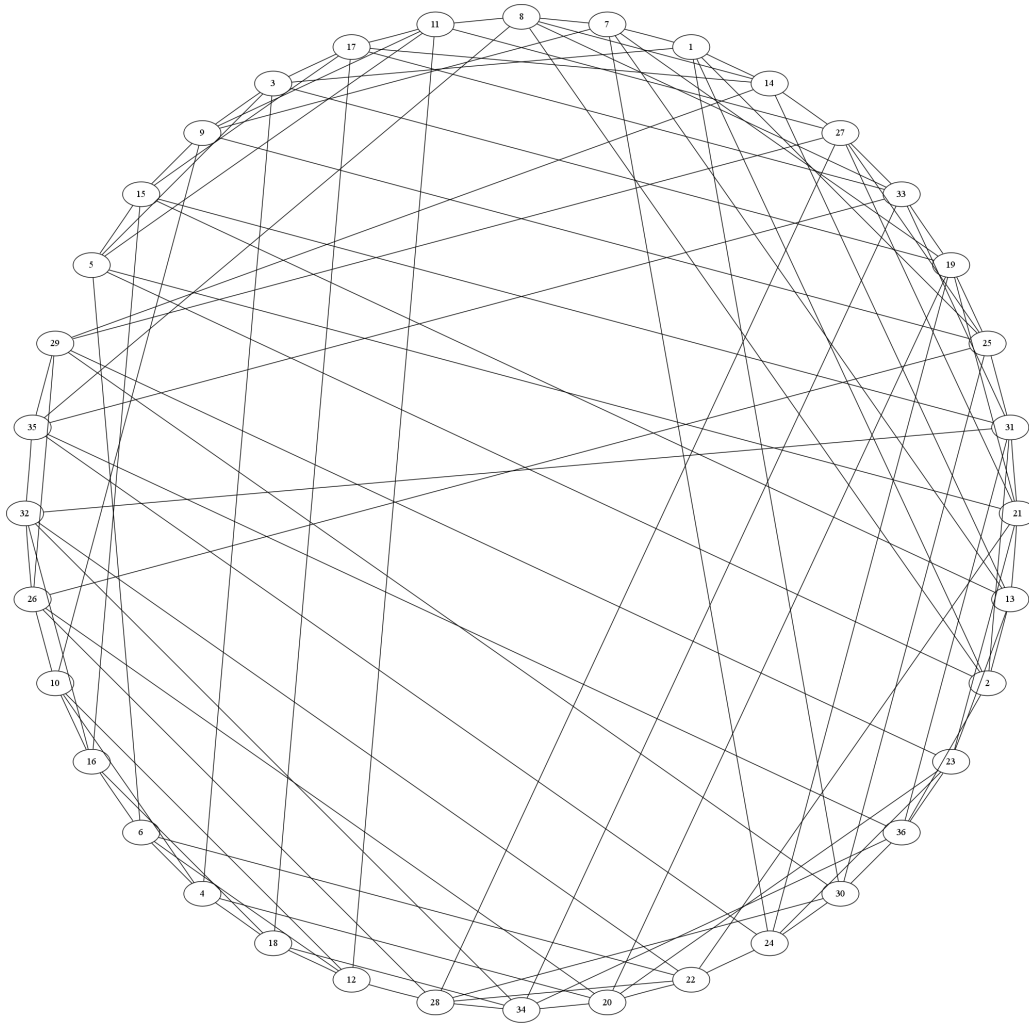
Figure 4.18: Switching graph for a 5 by 4 Nonogram of type 136

Figure 4.19: Switching graph for a 5 by 4 Nonogram of type 137
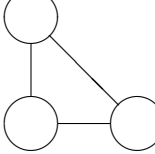
type	switching graph	occurrences
1		25 309 575
2		491 418
3		2 311 785
4		9 824
5		85 176
6		249 756
7		32 340
8...1915	various	291 945
1916	5-rooks problem	1

Figure 4.20: Some solution switching graphs in 5 by 5 Nonograms

Nonograms of types 8 through 1915 are not included. These types account for 291 945 different 5 by 5 Nonograms with a combined total of 1 507 043 solutions.

The single Nonogram of type 1916 is another example of the n-rooks problem. The Nonogram has $5! = 120$ solutions. In each solution, any two black cells can be used to perform a switch, meaning each solution is adjacent to $4 + 3 + 2 + 1 = 10$ other solutions. The maximum distance between any two solutions is 4.

4.4   Switching graphs and complexity

Having defined switching graphs for Nonograms, we propose to use these graphs as a measure of a Nonogram's complexity. The notion of a Nonogram's difficulty can be extended to the number of steps involved in finding the optimal partial solution, which is the full solution for uniquely solvable Nonograms. For Nonograms with multiple solutions, we are then left with a flexible set which can be colored in several different ways to construct all the different solutions of the Nonogram.

The number of solutions alone is not a satisfactory indicator of a Nonogram's complexity as some Nonograms have a small number of solutions which are difficult to find and others, like n-rooks Nonograms, have a great many solutions which can be determined very easily.

Switching graphs can be used as is, but it is not easy to compare one switching graph to another and say which indicates more complexity. As such, one might define some function to compute a numerical complexity given a switching graph. Our first proposal for such a function is

$$Complexity(G) = (|G| - 1) / \log_2(|Aut(G)| + 1)$$

where $Aut(G)$ is the automorphism group of switching graph G. In this function, the number of solutions for a given Nonogram forms the basis of its complexity, but Nonograms with highly symmetric switching graphs will have significantly lower complexities. For example, Nonograms which resemble the n-rooks problem generally have many solutions, but those solutions are highly regular and easy to generate, which is reflected in highly symmetric switching graphs.

Figure 4.23 shows the graph size, the automorphism group, the size of the automorphism group and the complexity for the different types of 4 by

4 Nonograms as listed in Figures 4.6 and 4.7. Here G refers to the switching graph as shown in those figures.

The automorphism group $Aut(G)$ of a switching graph G is presented as a direct product of three types of groups: C_n is the cyclic group of order n, S_n is the symmetric group of degree n with order $n!$, and D_n is the dihedral group of order $2n$.

For example, take the switching graph of 4 by 4 Nonograms of type 11, repeated with numbered nodes in Figure 4.21. The nodes in this graph can be rotated into four different configurations without altering the structure of the graph. Furthermore, the graph can be reflected and the reflection can once again be rotated into four different configurations without altering the structure of the graph. No further symmetries exist in this graph, which means the automorphism group has order 8.
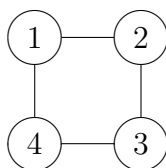

Figure 4.21: Switching graph for a 4 by 4 Nonogram of type 11

Another example, the switching graph of 4 by 4 Nonograms of type 15, is repeated with numbered nodes in Figure 4.22. This is the complete bipartite graph $K_{3,3}$, a graph with two partitions of three nodes each where each node is connected to all nodes in the other partition and no nodes in its own partition. Using the numbering in Figure 4.22, each odd numbered node is connected to all even numbered nodes and there are no edges between two nodes which are both odd numbered or both even numbered.

As each node in a given partition is only connected to all nodes in the other partition, nodes within the same partition can be switched without altering the structure of the graph. There are $|S_3| = 6$ possible configurations of the odd numbered nodes and $|S_3| = 6$ of the even numbered nodes. As these configurations are completely independent from one another, switching nodes within their respective partitions allows for 36 configurations with the same graph structure. Furthermore, the graph can be reflected along a fixed axis in any of these configurations, essentially switching the two partitions, which means the automorphism group of this graph has order 72.
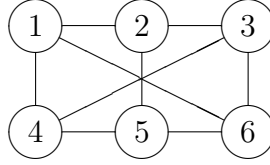
Figure 4.22: Switching graph for a 4 by 4 Nonogram of type 15

type	$|G|$	$Aut(G)$	$|Aut(G)|$	Complexity
1	1	C_1	1	0.000
2	2	C_2	2	0.631
3	2	C_2	2	0.631
4	3	S_3	6	0.712
5	3	C_2	2	1.262
6	3	C_2	2	1.262
7	4	C_2	2	1.893
8	4	$C_2 \times C_2 \times C_2$	8	0.946
9	4	C_2	2	1.893
10	4	S_3	6	1.069
11	4	D_4	8	0.946
12	5	C_2	2	2.524
13	6	C_2	2	3.155
14	6	$C_2 \times C_2$	4	2.153
15	6	$C_2 \times S_3 \times S_3$	72	0.808
16	8	$C_2 \times C_2 \times C_2$	8	2.208
17	9	S_3	6	2.850
18	24	-	-	-

Figure 4.23: Automorphism groups and complexities of 4 by 4 Nonograms

The table in Figure 4.23 shows some clear examples of Nonogram types which have a very low complexity compared to the number of solutions because their switching graphs have large automorphism groups. Type 15 stands out in particular. These Nonograms are essentially the same as the 3-rooks problem, which has six solutions which are very easily generated. Unfortunately, we were not able to determine the structure of the automorphism group for Nonograms of type 18. Nonograms of type 13 also have six solutions, but these solutions appear to be much less structured. Types 12, 13 and 17 stand out as having small automorphism groups for the number of valid solutions that exist for those Nonograms.

Chapter 5

Conclusions

We have presented an improved method to construct uniquely solvable Nonograms which resemble grayscale images. This method is flexible and provides many options which can be used to improve performance in several ways, including faster runtimes and increased similarity to the grayscale image. We have also presented ways to generate multiple Nonograms, different from one another, but still resembling the same grayscale image and often having varying difficulty levels. The precise options and parameters used to generate Nonograms should be adjusted to the image used as input as certain combinations of parameters work very well for some images and badly for others. It is particularly important to choose an initialization method that is suitable for the input image. Not all images can be turned into uniquely solvable Nonograms that are recognizable as being based on the images used.

We have also described how switching components can be used to transform multiple solutions of the same Nonogram into one another and how they can be combined into generalized switching components. We have used switching components to calculate the distance between Nonogram solutions and to build switching graphs, which represent the structure of Nonogram solution groups. These switching graphs can be used as a measure of the complexity of a Nonogram and we have proposed a way to calculate a Nonogram's complexity by combining the number of solutions with the order of the automorphism group of the Nonogram's switching graph.

5.1 Future Work

There is still much room for new ideas and research into the automated construction of Nonograms. The method described in this paper can generate a great number of Nonograms and select some of varying difficulty, but there may be ways to direct the difficulty of a Nonogram as it is being constructed.

Further research into generalized switching components could prove very interesting. We have described generalized switching components and discussed how they can be found, but it might also be possible to identify generalized switching components without first generating all solutions of a Nonogram.

The measure of a Nonogram's complexity defined in this thesis is but a first attempt. We have not used this definition much, nor have we compared the complexity of various Nonograms with the problems that Nonogram solvers run into when trying to solve those Nonograms.

Everything discussed in this thesis may also be applied to Nonogram variations. For example, there are Nonograms made up of triangles instead of squares, with line descriptions provided for three different directions. There are also colored Nonograms, where only groups of the same color need to be separated from one another by white pixels. In such Nonograms, a cell is no longer a binary entity, which may have a significant impact on both solving and construction algorithms. Additionally, having multiple colors available changes the way in which a Nonogram's solution may resemble an input image.

Bibliography

[BHKP09] K.J. Batenburg, S.J. 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.

[BK04] 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.

[BK09] K.J. Batenburg and W.A. Kosters. Solving Nonograms by combining relaxations. In *Advances in Combinatorial Image Analysis, Proceedings 12th International Workshop on Combinatorial Image Analysis*, volume 42, pages 1672–1683. Elsevier, 2009.

[Buz08] T.M. Buzug. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*. Springer Berlin/Heidelberg, 2008.

[Her09] G.T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Springer London, 2009.

[HK99] G.T. Herman and A. Kuba. *Discrete Tomography: Foundations, Algorithms, and Applications*. Birkhäuser Boston, 1999.

[HK07] G.T. Herman and A. Kuba. *Advances in Discrete Tomography and its Applications*. Birkhäuser Boston, 2007.

[Kos08] W.A. Kosters. Website Nonogram construction. `http://www.liacs.nl/~kosters/nono/`, 2008.

[Rys57] H.J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian Journal of Mathematics*, 9:371–377, 1957.