Universiteit Leiden
Opleiding Informatica

An Automatised Process

for Constructing Recognisable Nonograms

Name: Lai-yee Liu

Date: 28/08/2014

1st supervisor: Walter Kosters
2nd supervisor: Michael Lew

BACHELOR THESIS

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

Contents

Abstract

A Nonogram is a type of logic puzzle where the goal is to fill in an empty, rectangular grid based on line descriptions to reveal a meaningful image consisting of black and white cells. Constructing large, puzzle book Nonograms manually is an effort demanding process and not much research exists concerning how to automatise it. It is challenging as Nonograms have many subjective qualities to them such as difficulty, recognisability, and appealingness, which requires a machine to emulate human-like vision and creativity.

The program in this project is based on an existing algorithm that transforms an input image into a uniquely solvable Nonogram. The major steps consist of first reducing the input image into a binary image of requested size called the base puzzle (which entails the use of image processing techniques) and then ensuring that the base puzzle becomes a uniquely solvable Nonogram by iteratively adding black cells to it. The proposed method puts emphasis on recognisability by using edge detection to yield the base puzzle, which has benefits over the original thresholding approach. Then for the prevalent runtime issues for the second step, the new method reduces the search scope and promotes appealingness in the Nonogram by blacklisting some candidates. Tuneable parameters are now replaced with predefined configurations to make the method more suited as an automatised approach. Results from test inputs consisting of photos of human faces, vehicles, and animals, cartoon style images, and pixel art show that it is faster than the original program and most of the resulting Nonograms were of good quality.

Acknowledgements

I especially want to thank my supervisors Walter Kosters and Michael Lew for all the guidance and support they provided me on this Bachelorproject. Also special thanks go to family and friends for being always enthusiastic but critical at evaluating the Nonogram results.

1 Introduction

A Nonogram is a type of logic puzzle introduced in the eighties in Japan and has continued to remain popular throughout the world even today.
The goal is to fill in an empty, rectangular $m \times n$ grid based on line descriptions, with m denoting the amount of rows and n the amount of columns. There are two available colours for a cell: black or white. A line description, either for a row or column, consists of one or more elements called clues. The value of a clue, in the form of a non-negative integer, indicates the length of a consecutive black segment (also called a chunk) in that specific line and the amount of black segments can be determined from the number of clues in that line description. The order of the black segments has to adhere to the sequence of clues. For rows this is read from left to right and for columns this is read from top to bottom. Black chunks are always separated by one or more white pixels and there may be additional white pixels before the first black chunk and additional white pixels after the last black chunk. The solution to a puzzle is correct when the grid is consistent for all row and column descriptions.
Take for example the description (2, 1) in the first row of Figure 1. There are two clues in this row description, which means that there should be two black segments in this row. The first segment consists of two black pixels and the second segment consists of only one black pixel. Figure 1 shows a complete puzzle and its unique solution.

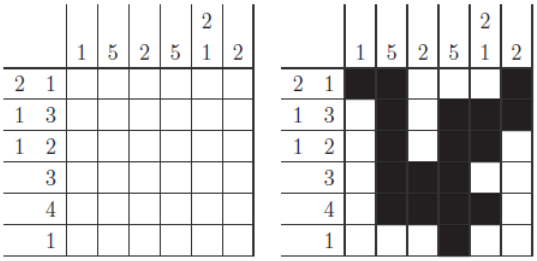

Figure 1: Puzzle on the left and its solution on the right [4].

There are variations on Nonograms, such as puzzles including more than two colours, but this project only focuses on the default black-and-white variation. Therefore for brevity the term Nonogram as used in this thesis only refers to these, while strictly speaking it should not be. Nonograms are also known by other names, including Japanese puzzle, Hanjie, griddler, Picross, Pic-A-Pix, and paint-by-numbers puzzle.

1.1 Puzzle Book Nonogram Designs

To encourage human players to actually finish solving the puzzles, Nonograms found in puzzle books, videogames or puzzle websites describe a meaningful or recognisable image. Figure 2, Figure 3, Figure 4, and Figure 5 taken from [1] showcase the level of expressivity achievable by a Nonogram from a certain dimension.
The size of a Nonogram grid tends to be a multiple of 5 for height times the multiple of 5 for width. This makes it easier for human solvers to count the cells.
Small nonograms of sizes of size 5×5 (Figure 2) have a minimalistic, abstract representation of the subject. In combination with the title of the Nonogram the content can still easily be figured out. Nonograms of size 10×10 (Figure 3) are already able to express content that can be recognised without the title's aid and some 20×20 Nonograms (Figure 4) even look very pleasant as stand-alone pictures. There is no formal definition for a "large" Nonogram, but in general a Nonogram that has a height or width exceeding 40 cells and a total cell count of over 1000 is considered large. Thus all the nonograms shown in Figure 5 fall in this size

classification. Thanks to the "huge" dimensions, more details and objects can be depicted in one image, which allows a Nonogram artist to come up with very picturesque designs. Juxtaposing Figure 4e, Figure 4f, Figure 5d, Figure 5e, and Figure 5f shows the differences between Nonograms of varying dimensions but with similar theme. The lone butterfly in Figure 5f looks almost like a real-life butterfly in terms of shape compared to the butterfly in Figure 4f. A similar observation can be made between Figure 5d and Figure 4e. While the butterfly designs themselves in Figure 5e have approximately the same size in pixels as the butterfly in Figure 4e, due the extra space it can include a surrounding or background with extra objects like flowers.

Figure 2: Solutions of Nonograms of size 5×5 (from left to right): (a) "Trampoline", (b) "Order in Court!", (c) "Ambulance", (d) "Dragonfly", and (e) "Running Down the Stairs".

Figure 3: Solutions of Nonograms of size 10×10 (from left to right): (a) "Water", (b) "Shopping", (c) "Bonsai Tree", (d) "Cow", and (e) "Dino".

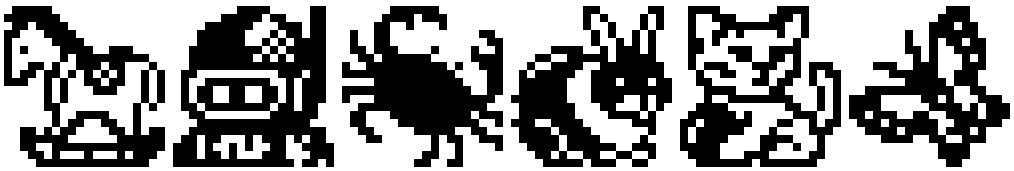

Figure 4: Solutions of Nonograms of size 20×20 (from left to right): (a) "Toy", (b) "Scuba Diver", (c) "Crab", (d) "Bunny Rabbit", (e) "Where's My Other Ball?", and (f) "Butterfly".

Figure 5: Solutions of large Nonograms (from left to right): (a) "In the Forest", (b) "Skyline", (c) "Madonna", (d) "Cat", (e) "Butterflies", and (f) "Butterfly". Figures a-e have size 50×50 and Figure f has size 45×45.

1.2 Research Problem

Designing Nonograms is a time and effort demanding task, especially when quotas and deadlines for weekly publications need to be met. The most important criterion for a puzzle book Nonogram is that it has to be uniquely solvable to ensure that the solution is always the image the creator intended. Despite the verification, to check whether the aforementioned applies, can be done in milliseconds time by many existing solvers, the solvability criterion limits the amount of grid configurations greatly and it is not easy for a human artist to find a grid configuration that is both uniquely solvable and also realises what he or she creatively wants to express. Section 2 focuses on the properties of puzzle book Nonograms.

It is beneficial to be able to automate this process where loads of Nonograms can be generated in batch, yet there is not a lot of research on this topic. So far there are only two known available programs: [13] and [10], where the user can supply a reference input image which is automatically converted into a binarized image as an output with the constraint that the black cell placements satisfy a uniquely solvable Nonogram. This project is an expansion on the existing source code from Sjoerd Henstra. For a more detailed explanation on all the features in the program, see [10]. Whilst *BasicNonogramGenerator* already contains all the features to automatically construct uniquely solvable Nonograms, the two main reasons why this project is initiated to follow up on that one are:

- **Parameter optimization**: Allowing a user to manually configure parameter values of a program that is practically a black box in their perspective would end up into a trial-and-error process where the parameters are adjusted randomly to test whether the result has improved to be more to their liking. Furthermore, the trivial Nonogram solution of a grid with either only white cells or black cells is always uniquely solvable and derivable from any input image, and the new software optimises the parameters in such way that the resemblance to the original image is of highest priority to avoid this phenomenon.
- **Speed-up**: The original program can get stuck in a state where it requires a lot of computations, so that there are situations where the user have to wait hours before an output is returned. This thesis mentions methods to circumvent the long wait.

For clarity, the original program will be referred to as *BasicNonogramGenerator* and the modified version proposed in this thesis as *ModifiedNonogramGenerator*. Figure 6 illustrates the program flow of *BasicNonogramGenerator*. Section 3 describes the Image Processing techniques that are used for the conversion from a user-submitted input image to the *base puzzle*. Section 4.1.1 covers the algorithms that guarantee a uniquely solvable Nonogram within reasonable time. The solvers used for verification are also discussed in Section 4.1.1.

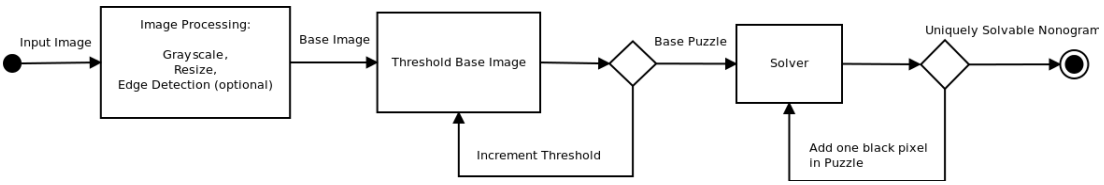

Figure 6: System flow

Ideally, the program can cover all kinds of input images but in practice this is not the case as the conversion to resembling Nonograms works better for some images than for others. Section 5 describes some test case studies where categories of image styles and contents are evaluated. As seen from Figure 5, large Nonogram grids allow expressivity of realistic-looking objects and with the assumption that the conversion works as well the other way round when using photos as input, this project only focuses on creating large Nonograms. Section 6 concludes this thesis.
This thesis is the result of a bachelor project at Universiteit Leiden, and is supervised by dr. W.A. Kosters and dr. M. Lew.

2 Puzzle Book Nonogram Properties

The difficulty of the problem lies in the craftsmanship of constructing a Nonogram where both the solving procedure and the solution itself provides a feeling of reward for the human solver. Hence the puzzle should provide a challenge and should not be too easy, plus the solved picture has to look nice and recognisable. Optimising only one criterion is insufficient; this yields Nonograms that can be challenging to solve but with meaningless images, or the complete opposite situation where the image is recognisable while the puzzle itself is boring to solve.
This section illustrates some properties for puzzle book Nonograms and rating functions that calculate how that Nonogram performs on that property. Rating functions are especially useful for sorting multiple solutions or introducing heuristic algorithms such as genetic algorithms (GAs) to solve a problem. The project does not utilize the rating functions yet for the above purposes, because they can only be used when they are proven to make judgements similar to what is desired; for instance emulating an average person. Besides, the human factor itself brings a lot of subjectivity and unpredictability is inevitable for ratings based from human judgement.

The list of properties is as follows:

1. **Uniquely Solvable**: The Nonogram must be uniquely solvable.
2. **Difficulty**: Indicates the degree of challenge provided by the Nonogram.
3. **Recognisability**: Recognisability of the Nonogram image.
4. **Appealingness**: How appealing the Nonogram looks.

2.1 Difficulty

The two difficulty classes for Nonograms are as follows [4]:

- **Simple**: Simple Nonograms can be solved through attaining new information each time from single line descriptions, which is known as SLS (Single Line Solving).
- **Hard**: Hard Nonograms cannot be solved through SLS and has to rely on "special" logic techniques. Examples of these special techniques are relying on multiple lines for new clues or guessing the colour of a cell and keep on solving until one arrives at a contradiction which then calls for backtracking.

The Nonograms found in puzzle books are usually solvable by SLS, nevertheless the hard puzzles are introduced to allow for an even greater challenge.

Different people have their own definition of what they find challenging. A relatively suited a posteriori measurement of difficulty is the average time taken by human players to solve the puzzle. This takes human error (extending time in need to fix mistakes) into consideration. Of course it is hardly possible to obtain this data during the designing process.
Consequently, the amount of work, or the complexity, to solve a Nonogram is calculated to serve as its difficulty measure. The complexity is defined as a positive number that describes the number of required sweeps (either horizontal or vertical). A horizontal sweep is going through each single row to acquire new cell colourings from row descriptions, and a vertical sweep is going through each single column respectively. Simple Nonograms can be solved through constantly alternating between horizontal and vertical sweeps, but the complexity of hard Nonograms cannot be measured this way and a suitable difficulty measure is yet to be formulized for that class of puzzles.
The average amount of new cell colourings gained per sweep is lower when the complexity is high, and it occurs more frequently that very few new pixels are obtained in a sweep when

the complexity is significantly high. For a human solver this means that more time is spent on searching these specific pixels to proceed and since they are so few it is easier to miss them. One of the drawbacks of this complexity value is that it does not take into account that humans are prone to make errors for certain types of line descriptions, such as where it is easy to miscount when there is a long sequence of the same clue values, like (1, 1, 1, 1, 1, 1). Secondly, the complexity corresponds to the SLS method, whereas humans can apply a mix of different solving techniques. A commonly used strategy is educated guessing where the colour of an unknown cell is predicted from perceiving reoccurring patterns, symmetry, or recognising the content of the image. We observed from comments on [1] that Nonograms are considered unchallenging when the image is too predictable.

2.2 Recognisability

The recognisability of a Nonogram is the ability to express the important features of object(s) depicted in input image in some way in the Nonogram, so that a human viewer can recognise what object(s) the image is depicting. Humans are capable of recognising abstractions, for example a minimum requirement to describe a face is at least able to depict shapes in the eyes, nose and mouth. The location of the features and the formation they form in the image matters more than the actual shape of the features as seen in Figure 7.

Figure 7: (a) input image of Turing; (b) *base image* of Turing; (c) uniquely solvable Nonogram of Turing

Converting an input image into a uniquely solvable Nonogram entails information loss in almost all the cases, unless the initial input image is a binary image that already satisfies the constraint of a uniquely solvable Nonogram. For instance a binary or rescaled image cannot be converted back to the state where pixels have their original intensity or colour. Taking into consideration that the recognisability is bound to turn worse, the user has to pick a recognisable input image to minimize the impact. Contradictory as it sounds, the input image may not always be a good representation for the content or subject which needs to be portrayed. Factors that can hinder the recognisability can be unusual perspective, noisy surroundings, or depicting characteristics atypical for the subject. Figure 8a is a photograph of the Eiffel Tower in Paris taken from a frog's view perspective. Due the odd angle, the famous characteristics one would associate with the tower are not straightforwardly visible, and spacial awareness is required to recognise the shape of the Eiffel Tower. Figure 8b depicts Disney Land Paris with a horse tram in the foreground that detracts all attention from the trademark Disney Castle in the background. Figure 8c depicts Boo the Dog who is a Pomeranian, but he resembles a teddy bear more than a traditional dog. It is a recognisable image of "Boo the Dog", but not of the subject dog.

Two types of resemblance are distinguished to illustrate the program's capabilities:

- Nonogram's resemblance to the subject that needs to be portrayed
- Nonogram's resemblance in contrast to the *base image* (definition of *base image* can be found in Section 3)

Figure 8: Unusual perspective: Eiffel Tower (a); noisy surroundings: Disney Land Paris (b); characteristics atypical to subject: Boo the Dog (c)

Neither *BasicNonogramGenerator* nor *ModifiedNonogramGenerator* possess any understanding on the context of an image, therefore it is impossible to measure the first type of resemblance. To the contrary, the second type of resemblance can be theoretically quantified through comparing the *base image* and the output Nonogram, however we have not yet found a resemblance function that is useful in emulating human perception which works in complex ways.

3 Image Processing

It is almost never the case that an arbitrary given input image is immediately in a suitable Nonogram puzzle format. This section first provides a description on bitmap images in general in Subsection 3.1, which is then followed by several subsections about the image processing operations that are used to pre-process an input image for this Nonogram generating problem.
The image processing operations listed in order of application are as follows:

1. Greyscaling (Subsection 3.2)
2. Rescaling (Subsection 3.3)
3. Edge Detection (Subsection 3.5)
4. Thresholding (Subsection 3.4)

Thresholding is presented earlier than the edge detection, because most Nonogram generating programs (including *BasicNonogramGenerator*) utilize thresholding as a part of the main algorithm while edge detection is either an optional parameter or is not included at all. This thesis proposes that the edge detection step is vital for a recognisable and appealing Nonogram by showing first what results look like if only image processing steps 1, 2, and 4 are applied (without edge detection). As shown later, applying edge detection before the thresholding step enhances the recognisability and appealingness in *base puzzle* design.

The input image undergoes two stages; first the *base image* stage, and then the *base puzzle* stage. Figure 9 illustrates all the used image processing operations in a scheme.

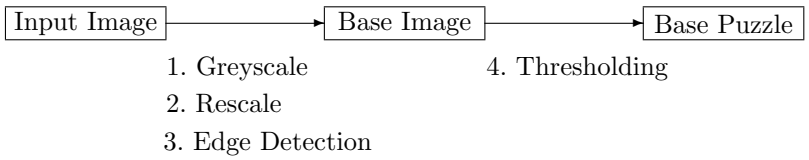

Figure 9: An overview of the image processing operators that are used in this program

The *base image* stage has the following characteristics:

- The resulting *base image* size matches the requested Nonogram size.
- An RGB colour image has 3 channels. The *base image* contains only one, namely the grey intensity channel.

The *base puzzle* stage has the following characteristics:

- The *base puzzle* size matches the requested Nonogram size.
- The *base puzzle* is binary and contains only black and white pixels.

The *base image* serves as a reference of what the original input image had looked like. We do not use the original input image as a reference, because it is much simpler and straightforward to do a comparison of pixel values in the *base image* and *base puzzle* when the sizes of the images are the same.
A *base puzzle* already has the format of a black-and-white Nonogram, because such an image can be described with Nonogram descriptors. However, it is most likely not uniquely solvable yet. The further processing of the *base puzzle* in a process we call `Adapt` is covered in the next section.

3.1 Bitmap Images

For this project we work with input images that are in 8-bit bitmap formats (for example `.png` and `.jpg`). A bitmap image consists of pixels in a $x \times y$ grid. A pixel value has to be an integer within the range from 0 to $2^b - 1$, with b denoting the bit size (in this case it is $2^8 - 1 = 255$). This is a discrete representation.
The whole representation of a grid and pixel values being integers allows a monitor or screen to project the image. However, an image should actually be interpreted as a continuous 2D signal $f(x, y)$. A bitmap image is a subsample of this function where only a finite amount of points are picked from this function to be represented as pixels.
Therefore between two pixels in a bitmap image there is technically no data, but it should be treated as if it is a continuous representation with the sections between the pixels having unknown values. Interpolation is used to estimate what value could have been between the pixels based on the data of neighbouring pixels.

3.2 Greyscaling

Colour images have several channels to represent colour. In the RGB colour model, there are three channels: the red channel, the blue channel, and the green channel. Each colour pixel is composed of three values and these individual values express their intensity for their respective colour channel. At intensity 0 a pixel is completely black and at 255 it is the maximum hue of that channel. There are also other colour models such as HSV, but the ranges in these channels are not 0 to 255.
To determine what value a Nonogram pixel should receive, the three values should be mapped into one value first. By greyscaling an image, only one channel is still present, and the values in this channel represent grey intensity where colours cannot be expressed anymore. At intensity 0 a pixel is completely black and at 255 it is white.
It should be noted that colour images can depict grey pixels which makes it impossible to tell from a black-and-white image whether it has one channel or multiple channels. Therefore greyscaling is always part of the process to ensure that there is only one channel.

3.3 Rescaling

Rescaling an image is to change the image size of an input so that it corresponds to the requested Nonogram size. Downscaling an input image is more common, since Nonogram sizes tend to be of extremely low resolution so that the input image is far too large. Interpolation filters in `ImageMagick` have been tested and there is not an interpolation method that stands out as the most suited for this problem, however there are some filters that are unsuitable: for example `cubic` creates images that are too blurry and filters with no smoothing effect can break up thin lines into disjointed pixels (for example `point` (nearest neighbour)). *BasicNonogramGenerator* uses `bartlett` but it could have been interchanged with another interpolation filter.

3.4 Thresholding

To turn a greyscale image into a binary image with only black and white pixels, thresholding is applied. Low intensity pixels in the *base image* are turned into black pixels (value 1) and high intensity pixels are turned into white pixels (value 0) and what is considered low or high is separated by the threshold value. All pixels of the *base image* are compared to the threshold value and the value the *base puzzle* will receive for that position is determined by:

$$BasePuzzle_{ij} = \begin{cases} 1 & \text{if } BaseImage_{ij} \leq \text{threshold value} \\ 0 & \text{if } BaseImage_{ij} > \text{threshold value} \end{cases}$$

3.4.1 Threshold Algorithms in Original Program

BasicNonogramGenerator proposes two algorithms that determines a threshold value for a provided *base image*, which are as follows:

1. `chooseSolvable`: The threshold value is the first encountered threshold value in the range [`min_threshold`, 255], that creates a uniquely solvable Nonogram. The algorithm begins at thresholding the base image with the `min_threshold`, but as long the generated base puzzle is not uniquely solvable yet, the applied threshold value is incremented by 1. There is always a trivial answer, since at threshold value 255 the base puzzle corresponds to a uniquely solvable all black Nonogram.

2. `chooseFill`: The threshold value is the first encountered threshold value in the range [`min_threshold`, 255] that satisfies a minimum requested black percentage for the *base puzzle*. Analogous to `chooseSolvable`, the increment is also by 1 when the criterion is not yet met. The original authors of the program recommend picking a percentage of 30 to 40. There is not a clear explanation why this is chosen, but for the most input images it seems to return a Nonogram that is somewhat recognisable.

3.4.2 Issues with Thresholding

First the issues of the two aforementioned thresholding algorithms are covered, and then the main issues with thresholding in relation with the Nonogram generating problem are covered.

Issues with `chooseSolvable` algorithm:

From testing, we came to the conclusion, that while `chooseSolvable` is an extremely fast method, it is not useful since the resulting Nonograms are often completely unrecognizable or uninteresting. For most input images, very few `base puzzles` are immediately uniquely solvable Nonograms unless the `base puzzle` contains extremely many black pixels or the opposite with extremely few white pixels.

Issues with `chooseFill` algorithm:

The `chooseFill` algorithm is significantly more useful than `chooseSolvable`, but still has its share of issues.

Fulfilling a requested minimum black percentage is problematic if it leads to a threshold where the black percentage ends up far higher than what we want. In the extreme case this turns all the pixels into black, which is quite common for line style drawings (with white background) which have a lower percentage of dark pixels than an average photograph. Therefore it is more intuitive to pick the threshold value that yields the percentage of black pixels in the *base puzzle* that has the minimal difference to the requested percentage instead of exceeding it:

$$|requestedPercentage - blackPixelsBasePuzzlePercentage| \rightarrow min \tag{1}$$

Using a constant black percentage value can be problem by itself. While 30 to 40 is the recommended percentage, there are of course exceptions where the image result is not recognisable. Questions can then be asked whether such an exceptional case can be detected and if it is the case, should it then be differentiated and get a different black percentage?

General Issues in Thresholding:

The issues that are mentioned next are not dependent on the thresholding algorithm that is used to find the threshold value.:

- Missing important details
- Portraying irrelevant details
- Unrecognisable silhouette

Missing important details
The primary use of thresholding is to segment an image, so that the important features are expressed as black pixels and the rest as white pixels in the binary image. In general the black pixels in a Nonogram are to denote the relevant details of an object on a white background (inverted Nonograms do exist but are far less common). Given these two statements, thresholding appears to be an attractive tool to tackle the Nonogram construction problem when the general purpose of thresholding matches the requirements imposed by the problem. However, thresholding works under the assumption that all the pixels that have a lower intensity than the threshold value are part of a relevant detail, but in practice parts from a scene or objects in an input image that have a lower intensity do not imply that they are more relevant details than higher intensity parts. Due to this, important details can be missed.

Portraying irrelevant details
At times it can be even the case that low intensity pixels in the *base image* are preferred not to be included, such as the background that happens to have a lower intensity than the foreground features. This leads to including the background, which is considered an irrelevant detail in the image. In most cases these dark background pixels come in a large frequency that satisfies the requested black percentage criteria, so that features of the object itself are not depicted at all.

Unrecognisable silhouette
Figure 10 depicts a cartoon character in an image where there is a clear separation between background and foreground, as the background consists of white pixels of intensity 255 and the object itself of lower intensities (except the feet). Especially due to the fact that this image does not have a great variation in intensities and where the light grey pixels of the face, head, and body are in majority, picking the default recommended black percentage of 30 turns almost the entire cartoon character to black. This creates a silhouette-like effect, where a silhouette is defined to be a shape that depicts the outer perimeter of an object with all the pixels within the object filled in as black (high intensity parts within object might be expressed as white holes though). Because this cartoon character is not a universally recognised character or object, its silhouette makes it extremely hard to make out what the image represents. Silhouettes are not inherently unrecognisable, and an example of a recognisable silhouette of a bicycle is found in Figure **??**.

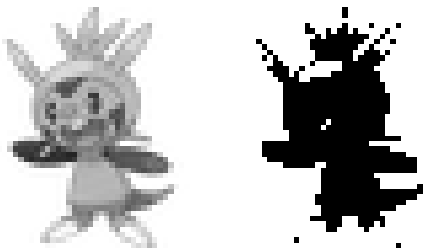

Figure 10: A *base image* of size 45 × 45 is depicted on the left and the corresponding *base puzzle* with minimum black percentage set on 30% is depicted on the right.

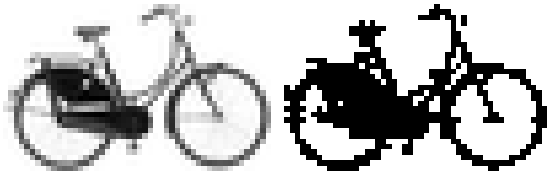

Figure 11: A *base image* of size 50 × 31 is depicted on the left and the corresponding *base puzzle* with minimum black percentage set on 30% is depicted on the right.

3.5 Edge Detection

We propose to consider edge detection a major step to acquire the *base puzzle* instead of leaving it optional and only applying thresholding on the *base image*. Edge detection is a way to extract contours from an image and this enables the possibility to create a line-art style abstraction of the *base image* for the *base puzzle*. Contours are defined as a line or set of lines enclosing or indicating the shape of an object in a sketch or diagram. To explain it further we make a distinction between 2D world and 3D world. When a scene presented in an image is viewed as if in a 2D world, a bounding contour encapsulates the parts in a scene that belong to the object(s) to segment them from the background. When viewing it as in a 3D world, an object itself has features that pop out towards the foreground or sink away in the background, and occluding contours found within the object are used to describe the physical shape of the object. For this context we define contours as including both the bounding contours and occluding contours.
First an explanation of how edge detection works is given in Section 3.5.1, which is followed by a comparison of different edge detection methods in Section 3.5.2 to examine which edge detection method is most suited to create the *base puzzle*. Finally the benefits of edge detection over thresholding is described in Subsection 3.5.5.

3.5.1 How Does Edge Detection Work?

A greyscale image is 2-dimensional and can be seen as a landscape where the intensity is the height. Edge detection makes use of convolution (which is explained below) to locate points within an image where there are sharp changes in gradient. The main idea is to compare the intermediate image that resulted from convolution, which we call a gradient map, with an edge threshold to decide which gradient changes are strong enough consider it belonging to an edge or not. The gradient map consists of pixels ranging from 0 to 255, which the higher the intensity of pixel the greater the probability that it is an edge pixel. The end-result is an edge map that is a binary image where black pixels denote edge pixels. This is not the normal convention, but features from edge contours should be black to adapt it to the Nonogram construction problem.

Convolution

Convolution is a mathematical operation on two functions $f(x)$ and $g(x)$ to produce a third function $h(x)$ that is a modified version of the original functions (see equation 2) [9]. In image processing convolution is often used to modify an image by weighting the signal of some input image with another signal to create a new image. The convolution operation is denoted with an asterisk symbol:

$$h(x) = f * g(x) = \int_{-\infty}^{\infty} f(u) \cdot g(x-u)\,\mathrm{d}u \tag{2}$$

It has been previously mentioned that bitmap images are discrete functions, so the convolution that is applied to bitmap images is a slightly adapted variant that works on discrete

formats, but the main idea is the same. An input image $f(x, y)$ is modified with a spatial convolution mask $g(x, y)$ (or as commonly called kernel) which yields an output image $h(x, y)$. The formal definition can be found in equation 3:

$$h(x, y) = f * g(x, y) = \sum_i \sum_j f(i, j) \cdot g(x - i, y - j) \tag{3}$$

In equation 3 it can be seen that convolution requires the inverted kernel. It is more common in literature to represent kernels in their pre-inverted form which we call $k(x, y)$. To approximate the output image $h(x, y)$, most image processing programs (*BasicNonogramGenerator* and *ModifiedNonogramGenerator* make use of `ImageMagick`) apply a small kernel. A kernel has one pixel that is specified as the pivot pixel. To generate the output image $h(x, y)$, each pixel value $f(x, y)$ of the input image is visited by sliding the kernel $k(x, y)$ on top of the input with the pivot pixel aligning the position of $f(x, y)$. Here, i and j represent the coordinates in kernel $k(x, y)$ with regard to the pivot pixel which has coordinate (0,0) in the kernel. The corresponding value for output pixel $h(x, y)$ is calculated by taking the sum of products of the kernel coefficients and input pixels that share the same positions:

$$h(x, y) = f * g(x, y) = \sum_i \sum_j f(x + i, y + j) \cdot g(i, j) \tag{4}$$

However according to convolution equation 4, the kernel can go out of bounds from the input image. There is no standard way to solve this problem and depending on what the implementation is to treat border cases in off-the-shelf programs, different results can be returned while using the exactly same kernel.

In the case of edge detection, a kernel defines the neighbourhood for locating the gradient changes. For the reason that it is not trivial to define an edge, many different edge detection methods exist and their effectiveness depends on the problem where they are applied to. Different images have different qualities which makes it difficult to take decisions based on assumptions and pick settings that work equally good for any arbitrary image. Since it is an automised system, what we do is experiment on different edge detection settings to find configurations that give generally good results for test cases. What is considered generally good is based on an a posteriori judgement by the author of this paper by taking recognisability and appealingness into consideration.

Edge Detection Phases

In general these are the four main phases for edge detection:

1. **Pre-processing: Smoothing**:
   A smoothing filter can be applied to an image to remove details and noise. The reason one might want to remove details is because not all details are equally important, for example a texture patterns on objects make an image overly complex. Known techniques include Gaussian Blur and Median Filter. These smoothing filters make the gradient landscape more streamlined, so that there are less abrupt discontinuities in the gradient. This translates to detecting less edge pixels.
   In the Nonogram construction problem we skip the smoothing step. Downscaling already provides an effect similar to smoothing because the newly assigned pixel intensities in the rescaled image are based on some average of the original pixels. Due to this, there are not many occurrences of discontinuities after applying downscaling. The amount of discontinuities that are present, though, is often sufficient to create edge maps that resembles contour line drawings. Applying a smoothing filter further eliminates the amount of detected edges, resulting into an unrecognisable and unappealing Nonogram with little details.

2. **Apply edge detection kernel**

3. **Apply edge thresholding**

4. **Post-processing: Edge Linking and Thinning**:
   The edge map may still contain misclassified edges or fails to detect edges, so cleaning it up with some post-processing yields a clearer edge map. Edge linking combines separated line segments that are close to each other into one longer edge. Edge thinning cleans up the edge elements by removing spurious edge pixels, so that edges are all 1 pixel thick.
   No special attention has been given to edge linking and edge thinning in our problem. Angled lines and curves of 1 pixel thick have their issues that are described in the next section, which is the main reason why edge thinning is not applied.

3.5.2 Edge Detection Operators

There are many types of edge detection methods, but they are mainly grouped into these two categories [7]:

1. **Gradient based edge detection**:
   Operator depends on first order differential and searches for the maximum or minimum peaks. Operators that fall in this category are Sobel, Prewitt, Scharr, and Roberts-Cross.

2. **Laplacian based edge detection**:
   Operator depends on second order differential and searches for zero-crossings. Operators that fall in this category are: Laplacian, DoG (Difference of Gaussians), LoG (Laplacian of Gaussians), or also called the Marr-Hildreth method, where Laplacian is combined with Gaussian blurring: an image is smoothed first, then the zero-crossings are found.

Canny edge detection [8], which is a combination of both first and second order derivative, is considered to be a popular edge detection technique in many general applications of machine vision. The popularity can be attributed to the major feature of canny which — unlike the old classical operators like Sobel or Laplacian — takes two thresholds: an upper threshold and lower threshold in the so-called hysteresis step. Every pixel with value above the upper threshold correspond to actual edges. The pixels with value between the lower threshold and upper threshold correspond to potential edges: whether they are actually part from edges is decided by a recursive process through checking whether they are connected to edge pixels by also taking into account the direction of an edge. Canny allows for higher accuracy than classical operators like Sobel or Laplacian in finding the actual edges and leaving out false positives if the parameters are set correctly.

The following edge detection operators were tested on input images and the results can be found in Section 3.5.4:

- Sobel
- Scharr
- Roberts-Cross
- Prewitt
- Laplacian 3×3 Diamond kernel
- Laplacian 9×9 Diamond kernel

- Laplacian 3×3 Block kernel
- Laplacian 9×9 Block kernel

  (Note that all of the kernels above were tested in `ImageMagick`)
- Canny (as implemented with the parameters configurations in a demo program from `OpenCV`)

Laplacian methods only use one kernel while Sobel, Scharr, Roberts-cross, and Prewitt rely on two kernels. The gradient map is a combination from the two intermediary results, where its pixel value $S(i, j)$ is calculated by:

$$S(i,j) = \sqrt{(horizontal\_kernel(i,j)^2 + vertical\_kernel(i,j)^2)}$$

where i denotes the row position and j denotes the column position. Also depending on whether the edges are depicted as high intensity in the resulting gradient map, an additional invert operation might be needed afterwards to ensure that contours are depicted in low intensity.

We have the following kernels:

−1	0	1
-2	0	2
−1	0	1

1	2	1
0	0	0
−1	-2	−1

Figure 12: Sobel: (a) horizontal kernel; (b) vertical kernel

3	0	-3
10	0	−10
3	0	-3

3	10	3
0	0	0
-3	−10	-3

Figure 13: Scharr: (a) horizontal kernel; (b) vertical kernel

1	0
0	−1

0	1
−1	0

Figure 14: Roberts-Cross: (a) horizontal kernel; (b) vertical kernel

−1	0	1
−1	0	1
−1	0	1

1	1	1
0	0	0
−1	−1	−1

Figure 15: Prewitt: (a) horizontal kernel; (b) vertical kernel

0	−1	0
−1	4	−1
0	−1	0

Figure 16: Laplacian 3×3 Diamond kernel

0	0	0	0	−1	0	0	0	0
0	0	0	−1	−1	−1	0	0	0
0	0	−1	−1	−1	−1	−1	0	0
0	−1	−1	−1	−1	−1	−1	−1	0
−1	−1	−1	−1	40	−1	−1	−1	−1
0	−1	−1	−1	−1	−1	−1	−1	0
0	0	−1	−1	−1	−1	−1	0	0
0	0	0	−1	−1	−1	0	0	0
0	0	0	0	−1	0	0	0	0

Figure 17: Laplacian 9×9 Diamond kernel

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure 18: Laplacian 3×3 Block kernel

−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	80	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1
−1	−1	−1	−1	−1	−1	−1	−1	−1

Figure 19: Laplacian 9×9 Block kernel

3.5.3 Edge Thresholding

After inverting the gradient map so that a low intensity corresponds to being a strong change in gradient, the following two methods decide which pixels in the gradient map are to be treated as edge pixels:

1. `fill30`
2. `reduceWeakEdges`

`fill30`:
This method applies `chooseFill` on the gradient map. We pick 30 as the desired black percentage to calculate the (edge) threshold.

`reduceWeakEdges`:
The `reduceWeakEdges` function is a simplified Hysteresis step that is inspired from the Canny method, except it does not take directions of edges into account. It uses two edge thresholds that have the following values:
`lowerEdgeThreshold` = 200
`upperEdgeThreshold` = 255
and pixels in the gradient map are instantiated in the following categories:

- **Strong edge pixel**: intensity is between 0 and `lowerEdgeThreshold`
- **Weak edge pixel**: intensity is between `lowerEdgeThreshold` and `upperEdgeThreshold`
- **Non-edge pixel**: intensity is higher or equal to `upperEdgeThreshold`

Strong edge pixels are automatically treated as edge pixels. Weak edge pixels can either become strong edge pixels or non-edge pixels. Strong edge pixels recursively turn their 8-connected weak edge pixel neighbours into strong edge pixels. Remaining weak edge pixels become non-edge pixels.
Both edge threshold values have been chosen in such a way that there is a high sensitivity so that even relatively weak gradients are considered to be edges. Since the `reduceWeakEdges` function does not allow control over the black percentage in the *base puzzle* and to arrive at a black percentage that is close to 30, we try to include as many pixels from the gradient map as we can, unless they belong to isolated, weak gradients which is mostly likely noise.

3.5.4 Edge Detection Methods Comparison

In this Section we first test the kernels that are described in Section 3.5.2 and pick the ones that create the most recognisable and appealing gradient map. For convenience, gradients are already depicted in their inverted forms. Then we compare the two edge thresholding methods that are described in 3.5.3.

Edge Detection Operator

Gradient maps differ when the edge kernel is applied to the inverted *base image*, therefore Figure 20 and Figure 21 depict both results. The captions underneath the Figures contain detailed comments. The Canny implementation only outputs the edge map and not the gradient map.
The Laplacian methods work quite well for the Nonogram construction problem, however the difference between the results from Laplacian 3×3 and Laplacian 9×9 are significant. They both have their advantages and disadvanges and are therefore both treated as useful edge detection operators.
Disjointed edges should not be common in Laplacian as it is described to theoretically bound to lead to closed contours around objects [9]. The edge maps from the block and diamond

Figure 20: Input image Audrey Hepburn of size 38×45
The first row consists of the gradient maps on the *base image*.
The second row consists of the gradient maps on the inverted *base image*.
The operators that are applied, as observed from left to right: Sobel, Scharr, Roberts-Cross, Prewitt, Laplacian Diamond 3×3, Laplacian Diamond 9×9, Laplacian Block 3×3, Laplacian Block 9×9, and Canny.
The last row consists of edge maps derived from the gradient maps in the second row when using `fill30`.

Figure 21: Input image Chespin of size 45×45
The first row consists of the gradient maps on the *base image*.
The second row consists of the gradient maps on the inverted *base image*.
The operators that are applied, as observed from left to right: Sobel, Scharr, Roberts-Cross, Prewitt, Laplacian Diamond 3×3, Laplacian Diamond 9×9, Laplacian Block 3×3, Laplacian Block 9×9, and Canny.
The last row consists of edge maps derived from the gradient maps in the second row when using `fill30`.

shaped kernels are almost similar for Laplacian 9×9, but there are differences in the cases for Laplacian 3×3. The block kernel is less sensitive to noise (as seen in the amount of speckles surrounding Chespin in Figure 21 in the edge map) and gradients also have a greater strength in Laplacian 3×3 Block Kernel than Laplacian 3×3 Diamond Kernel (as seen from pixels with lower intensity in gradient maps). For this reason we only use the block kernels for both Laplacian 3×3 and Laplacian 9×9.

Edge Detection Thresholding: `fill30` vs `reduceWeakEdges`

The Laplacian method is known to be very prone to detecting noise, which lead to the introduction of the `reduceWeakEdges` function. Figure 22 compares the *base puzzles* acquired from Laplacian Block Kernels with `fill30` and `reduceWeakEdges` respectively. It can be seen that the `reduceWeakEdges` function is succesful for filtering out the noise pixels for the edge map. The difference between using `fill30` and `reduceWeakEdges` on `edge9x9` edge map is barely visible, but `reduceWeakEdges` will still be used for both methods.

Figure 22: The first column depicts edge maps from applying `fill30` on Laplacian 3×3 Block Kernel.
The second column depicts edge maps from applying `reduceWeakEdges` on Laplacian 3×3 Block Kernel.
The third column depicts edge maps from applying `fill30` on Laplacian 9×9 Block Kernel.
The last column depicts edge maps from applying `reduceWeakEdges` on Laplacian 9×9 Block Kernel.

We come to the conclusion that the following edge detection methods are considered useful for creating *base puzzles*:

- `edge3×3`: Laplacian 3×3 Block Kernel with `reduceWeakEdges`
- `edge9×9`: Laplacian 9×9 Block Kernel with `reduceWeakEdges`

3.5.5 Benefits of Edge Detection

The major uses of edge detection for this problem are as follows:

1. Main use: Extract the shapes or contours of objects in the image (image recognisability)
2. Reduce the amount of black pixels for puzzle (puzzle difficulty)

Extract the shape contours of objects in the image (image recognisability)
Unlike with thresholding, edge detection is able to depict features of high intensity in the form of discernable contours. Compare Figure 23b with Figure 23c. It is still recognisable in Figure 23b that the image represents a car, because the defining features as two wheels at one side and the front of the car can be seen. But the windows and the roof of the car have entirely disappeared from Figure 23b. Figure 23c does a better job maintaining all of these details.
Pixels of intensity 255 can be considered non-edge pixels and since they come in high frequency, picking all pixels with intensity lower than 255 from a gradient map will still not create a *base puzzle* that is close to entirely black. Some examples are shown in Figure 24 and while not all of them are appealing, they are not unrecognisable either.

Reduce the amount of black pixels for puzzle (puzzle difficulty)
The puzzle complexity can be increased after applying edge detection to an image. Again, compare Figure 23b with Figure 23c. The Nonogram that was created without applying edge detection has a large surface of connected black pixels, which leads to descriptor clues that are long. The Nonogram for which edge detection we used has less of these huge black

surfaces, and instead there is a white space within the boundary of a contour, which corresponds to more descriptors of short(er) length in a line description. The descriptors are short because a contour tends to be not very thick.
Losing important feature details from thresholding does not only account for diminishing the recognisability, it can also make a Nonogram significantly easier to solve; especially when this involves the loss of a large portion of an image. With edge detection, there is a possibility that these features can be kept intact as edges and contribute to the black pixel count.

Figure 23: (a) Input Image of a car of size 68×45; (b) Created Nonogram without applying edge detection (complexity 13); (c) Created Nonogram after applying Laplacian 9×9 Block Kernel (complexity 63)

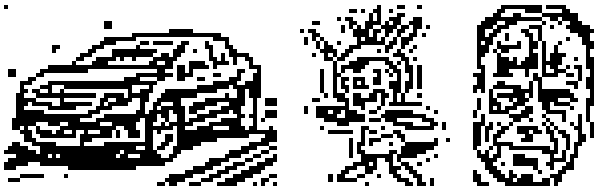

Figure 24: Results from considering all pixels with intensity lower than 255 to be edge contours in the gradient maps created from Laplacian 3×3 Block Kernel.

4 Solvers and Solvability

The previous chapter mentioned image processing techniques and after an input image has undergone image processing we have now arrived at a binary image we call the *base puzzle*. This chapter deals with further processing this base puzzle to make it uniquely solvable, which is briefly mentioned in Section 2 as one of the required properties of a puzzle book Nonogram. The term uniquely solvable will now be explained in more detail in this chapter. To determine whether a Nonogram is uniquely solvable, a solver is required and this chapter describes some of these. The used solver also determines the difficulty of the Nonogram.

4.1 Uniquely Solvable Nonograms

This section provides definitions related to the subject of the solvability of Nonograms.

Definition 1. Uniquely Solvable Nonogram:
A Nonogram is uniquely solvable if and only if there exists exactly one solution to the Nonogram.

Figure 25 depicts the smallest not uniquely solvable Nonogram, with two solutions.

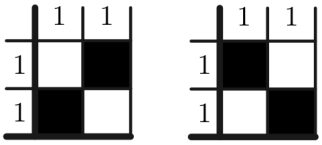

Figure 25: The smallest Nonogram with multiple solutions is for a 2 × 2 grid.

The problem of deciding whether another solution of a Nonogram exists, given a particular solution, is NP-hard [15].

4.1.1 Solvers

A solver is a strategy that tries to solve Nonograms.

Definition 2. Cell state
A cell can either take the state unknown or known, which are defined as follows:

- **known**: a cell in the partial solution for which the solver can decide that it should be exclusively black or white.
- **unknown**: a cell in the partial solution for which the solver cannot decide that it should be exclusively black or white.

Definition 3. Solution:
The Nonogram image contains no unknowns and the filled in cells have to adhere to the line descriptions.

Definition 4. Partial Solution:
The Nonogram image contains unknowns, but the cells that are already filled adhere to the line descriptions.

There are different algorithms to solve a given Nonogram puzzle and some of them have already been covered in Section 2 of this thesis. Some solvers are designed to retrieve all the possible solutions corresponding to a Nonogram description and some only retrieve one possible solution. A solver does not necessarily have to be able to solve a puzzle. The solvers that are mentioned next have the goal to retrieve one solution. We divide them into three categories for the purpose of elaborating why *BasicNonogramGenerator* (and *ModifiedNonogramGenerator*) uses a specific type of solver and not another:

1. **Heuristic, step-wise solver**:
   Heuristic, step-wise solvers only fill in cells whose colour can be decided with absolute certainty given the current state of the partial solution.
   Single Line Solving (SLS) is the most basic Nonogram solving technique that is recognised as the default way to solve Nonograms. This strategy determines the colour of the cells from descriptors in a single line with logical reasoning. The *SimpleSolver* of *BasicNonogramGenerator* is basically based on the SLS algorithm.
   Multiple Line Solving (MLS) can take multiple lines in consideration rather than one. The *FourSolver* that is used in the Nonogram generation program translates the dependency between 4 unknowns $u_{i,j}$, $u_{i+y,j}$, $u_{i,j+x}$, and $u_{i+x,j+y}$ where $i, y \in \{0, \ldots, m-1\}$, $j, x \in \{0, \ldots, n-1\}$, $y > 0$, $x > 0$, $i + y < m$, and $j + x < n$, into a 2-SAT Problem [5] when it cannot proceed with the *SimpleSolver*.
   Heuristic, step-wise solvers might not always solve a Nonogram and in this case the partial solution should include *unknowns*.

2. **Heuristic, step-wise solver that includes guessing**:
   This solving strategy is similar to the previous category, except it also allows for guessing cell values. Backtracking is usually applied on top of SLS when the solver cannot proceed further determining the values of unknowns. Backtracking assigns some value to an unknown value and then continues solving the Nonogram as long it does not lead to contradictions. If it does lead to contradictions, then all the steps after assigning a colour to the unknown pixel should be reverted and the unknown pixel should have the other value. While the principle does rely on logical reasoning, the initially guessed value is random. This strategy can be used when one solution needs to be acquired from a Nonogram that has one or more solutions. An implementation of a solver with backtracking is presented in [17].

3. **Non step-wise solver**:
   Unlike the other two categories, this covers the strategies that are not the traditional techniques applied by human solvers.
   Transforming the Nonogram problem to ILP is presented in [6] and [12]. The authors of [12] solve a Nonogram where the uniquely solvable property has to hold by utilising compressive sensing to solve the Nonogram encoded as an ILP.
   Genetic Algorithms try to find possible solutions by generating a completely filled-in Nonogram image and verify whether it is a solution, however they have issues with converging to a solution [2, 16]. The Taguchi-Based Genetic Algorithm yields more effective results than standard GA [14].
   A Depth First Search Algorithm generates all the possibilities for each row description and verifies which of these combinations are correct by evaluating the Nonogram image with the column descriptions [16], however it is not known to be particularly efficient.

When a Nonogram has multiple solutions, at least one pixel (four pixels in a stricter sense due to elementary switching components) of the Nonogram image can be assigned to be both black or white and still correspond to the same Nonogram description. The major purpose of the solver used in *BasicNonogramGenerator* is to confirm whether the *base puzzle* is uniquely solvable or not. Because it is an NP-hard problem to confirm whether there are multiple Nonogram solutions, *BasicNonogramGenerator* uses heuristic, step-wise solvers. These strategies only make logical decisions when it is absolutely certain that the assigned pixel values are correct (values cannot be rectified later, meaning that backtracking methods are ruled out). If a Nonogram is not uniquely solvable, it would lead to the algorithm getting stuck halfway the solving process when it encounters an unknown cell. Due to the existence of unknown cells in a partial solution, we now know that the Nonogram is not solvable with that solving algorithm.

While it may still be the case that a Nonogram is actually uniquely solvable a solver might be unable to solve it. So in the stricter sense, it still does not answer the question of whether it is uniquely solvable, but Nonograms that are able to be solved with heuristic, step-wise solvers are most definitely uniquely solvable. A Nonogram that is inherently not uniquely solvable regardless of what solver is used contains *switching components* [5]:

Definition 5. Switching components:
Switching components are cells where interchanging black and white cells results into in a different Nonogram with an identical description.

Simple solving techniques, for instance SLS, cover a smaller subset of uniquely solvable Nonograms than more advanced solver techniques as MLS (which also uses SLS). Therefore the used solver for generating a Nonogram also determines the difficulty of the created Nonogram.

4.2 How to Create a Uniquely Solvable Nonogram?

Section 4.1 provided the definition of uniquely solvable Nonograms and this section describes how a *base puzzle* is turned into a uniquely solvable Nonogram. Two algorithms are provided: the original algorithm *basicNonogramGenerator* [10] and a modification to that algorithm proposed in this thesis that puts emphasis on speed and image recognisability.

4.2.1 Original Adapt

`Adapt` is a function that is part of *BasicNonogramGenerator*. It chooses a white pixel from the current *base puzzle* to be changed into a black pixel with the aim to reduce the amount of unknown pixels. Pseudocode 1 describes the steps for the `Adapt` function.

U is the set of unknown pixels and is defined as a pair describing the x and y-coordinates. $\mathbb{L}$ is the set of generated Nonograms, which is used by *BasicNonogramGenerator* to generate multiple varying Nonograms from one input image.
There are three control parameters α, β, and γ; α determines the importance of reducing the amount of unknown pixels, β determines the importance of the created Nonogram to resemble the *base image* in terms of intensity values, and γ determines the importance of having variation among the generated Nonograms. The higher the value is for α, β, or γ, the more bias is put onto that criterion.

Algorithm 1 Original Adapt function

min $\leftarrow \infty$;
for $(i, j) \in U \cap$ such that $mathitbasePuzzle_{ij}$ is 0 **do**
  $basePuzzle_{ij} \leftarrow 1$; // change that pixel in basePuzzle to black
  fitness $\leftarrow \alpha \cdot unknownCount_{basePuzzle} + \beta \cdot baseImage_{ij} + \gamma \cdot \sum_{L \in \mathbb{L}} L_{ij}$;
  if fitness $<$ min **then**
    min $\leftarrow$ fitness;
    $(k, \ell) \leftarrow (i, j)$;
  end if
  $basePuzzle_{ij} \leftarrow 0$; // restore original basePuzzle
end for
$basePuzzle_{k\ell} \leftarrow 1$;
return *basePuzzle*;

The function call to `Adapt` turns exactly one white pixel into a black pixel. The loop that calls this function is stopped when the base puzzle had become a uniquely solvable Nonogram. There is always an end to this since a Nonogram consisting of entirely black pixels is

uniquely solvable.
It is sufficient to only check the white, unknown pixels, because unknowns always consist of at least one white cell [5] and adapting an unknown directly influences the amount of information that is available regarding at least four cells that are currently unknown to the solver, which hopefully makes it become uniquely solvable quicker with less iterations in the `Adapt` loop, although it is not guaranteed [10]. This is due to the fact that changing a white cell into a black one can introduce a higher amount of *unknowns* than before and a better decision (as in the priorities described by the α, β, and γ parameters) could have been found when transforming a white, known cell into black.
The `Adapt` function is very computation heavy, because for every single instance where a white pixel has been changed into black it calls the solver to calculate the new amount and location of unknown pixels. Yet at the same time it is a pivotal function for *BasicNonogramGenerator* and cannot be easily replaced with a different method when maintaining the structure of the program.
The used solvers on their own take less than a second to arrive at the conclusion whether a large Nonogram contains unknowns, so the issue lies more in the fact that the solver function is called so many times rather than that the solver is slow.
Take a hypothetical Nonogram of size 50×50 which contains 2500 pixels and 30% are black pixels, which accumulates to 750 black pixels. If all the pixels of the Nonograms are unknown at this state, then all 1750 white pixels can potentially become a black pixel and to check what the new state would be when changing one of them to black (and the rest stays white) would require 1750 solver calls. In the bad case — which is more frequent than not — changing one pixel into black in an iteration does not reduce the amount of unknowns, so the next iteration would require $s - 1$ solver calls, where s is the amount of solver calls in the previous iteration.
This example illustrates that it is not time efficient to treat all the unknown white pixels as black pixel candidates because the amount of required solver evaluations would then end up to be high (unless the unknown count is already low, but that is usually never the case). Hence the proposed method that is described in Section 4.2.2 has a similar approach to the `Adapt`, but it limits the scope of the search space to reduce the amount of solver calls.

4.2.2 Modified Adapt

The goal of the proposed method is to reduce the amount of Solver calls. To achieve this, not all the white pixels that are unknown are treated as potential candidates for turning into black. Then this requires some selection method to distinguish which pixels are to be candidates and which are not. A priority hierarchy is computed for each white pixel, that describes its rank of preference for turning into black.

The modified `Adapt` function work with priority levels:

1. **High Priority**
2. **Mid Priority**
3. **Low Priority**
4. **Blacklist Priority**

where the algorithm begins with the highest priority level. This means that it only treats pixels with that priority level tag as candidates in `Adapt`. The priority level that needs to be checked is lowered at the moment when all the unknown cells with this priority tag are already turned black. Analogous to `Adapt`, the stop criterion is when the *base puzzle* has become a uniquely solvable Nonogram. Note that the priority tags are distributed for white pixels, but the `Adapt` function requires the cell to both having a white value and being part of the unknown set, therefore for each cell that belongs to a certain priority level it requires

the check whether it is also an unknown cell. A cell can have more than one priority tag. The following relationship holds between the priority levels: *high priority* candidates $\subseteq$ *high priority* candidates $\subseteq$ *low priority candidates*.

High Priority

Given a block of cells of size 2×2 the *base image*, which we will refer to as $p_{i,j}$, $p_{i+1,j}$, $p_{i,j+1}$ and $p_{i+1,j+1}$, where $i = 0, \ldots, m-1$ and $j = 0, \ldots, n-1$: if either (1) $p_{i,j}$ and $p_{i+1,j+1}$ are white and the other two cells are black or (2) the colours the other way round, then the white cells in these conditions can potentially be *high priority* candidates. One of the two white cells is randomly picked as a *high priority* candidate. We do not pick them both, because this will lead to turning the 2×2 block into black. This happens, because the the priority level can only be lowered when the *high priority* candidates run out and it is rarely the case that a *base puzzle* with unknowns become uniquely solvable by only turning *high priority* candidates into black.
Diagonal line segments in a block become 4-connected when at least one of the two white cells are turned into black.
The *high priority* candidates are meant to reduce switching components that consist of adjacent cells. A possibility is to implement it in such a way that if it is a long line it checks on which of the two sides is more preferred to add the candidates, but that might overcomplicate things.

Mid Priority

First of all, *mid priority* candidates concern any white cells that are 4-connected to a black cell. This priority level is introduced to avoid having too many isolated lone black cells in the Nonogram. Even though adding a lone black cell may lower the count of unknowns drastically and may be a better choice than some cell adjacent to a black cell, these lone black cells are in general not considered as appealing for the image while cells that are adjacent to a black cell stand out less when added. It is implemented in such a way that *mid priority* cells are the 4-connected cells to black cells in the *base puzzle* before the algorithm has entered the `Adapt` loop.

Low Priority

All white cells in the base puzzle are in general to be treated to have the *low priority* tag. Before the priority levels were introduced, the original `Adapt` function evaluated all white unknown cells. This priority level is needed in case the higher levels fail to make the Nonogram uniquely solvable. It is not desirable to reach this level, because it reverts back to the original problem where too many candidates need to be evaluated. Fortunately, in most cases, very few exclusively *low priority* cells are needed to create a uniquely solvable Nonogram.

Blacklist Priority

For the image appealingness criteria, it is often desired that some cell should be preserved to stay white. A blacklist functionality had been implemented and will further be referred to in this paper under the name *blacklist priority*. It is technically one level lower than *low priority*, because when a *base puzzle* is still not uniquely solvable after going through *high* and *low* priority evaluations, then it checks the candidates which previously were not allowed to be modified.
The β parameter can be argued to already fulfil the purpose of preserving some cells to be white. It forces that a cell in a *base puzzle* which corresponds to a high intensity pixel in the

base image is less likely to be picked, however sometimes the pixels of a feature are not sufficiently light enough to be considered important. Given an example of a *base image* where there is some object and the background is white whereas the intensities of the object are lower than 255, so that when purely looking at the criterion controlled by the β parameter disregarding the other two parameters, it will deter the background pixels from turning into black while features of the object itself are at risk to be lost.
The proposal is to provide a *blacklist priority* tag to all the cells within a white polyomino (in the *base puzzle*) of size smaller than k pixels, where k is a constant discrete number. When $k \leq 0$ it is equal to not making use of cell blacklisting. Currently, k is set to 10.

Evaluation Function

For the evaluation of the fitness of changing a candidate into black, the original `Adapt` used the following formula as described in Section 4.2.1:

$$fitness = \alpha \cdot unknownCount_{basePuzzle} + \beta \cdot baseImage_{ij} + \gamma \cdot \sum_{L \in \mathbb{L}} L_{ij} \tag{5}$$

The third part with the γ can be left out when only creating one Nonogram. The second part with the β is not very useful for the edge detection approach. The edge map often already has intensities for pixels not matching its original intensity value in the *base image*, so this downplays the importance of strictly adhering to the intensities in the *base image* in the `Adapt` function. Leaving out the second part which controls the resemblance to the *input image* makes for a far easier fitness function, because then you do not need any of the α, β, and γ parameters. Counting the number of unknowns is a greedy approach with the sole goal to make the Nonogram uniquely solvable. If a new fitness value is equal to the currently best fitness value, then it has a probability of 0.5 to pick the new cell as the best.

4.2.3 Special erosion

Erosion is a morphological image processing operation that can be applied to binary images. It expands black pixels to their neighbours as according a kernel. The primary use is to thicken black lines, expand black shapes and remove details that consist of white pixels: such as filling up white holes in a black shape. It would appear more intuitive to describe this operation in Section 3 , however erosion is not actually applied in *ModifiedNonogramGenerator*, but an adapted version of erosion is used that has similar goals as erosion which we call `specialErode`.
Before describing the modified erosion, an explanation is given of the benefits of applying erosion on a *base puzzle*. We mentioned in Section 4.2.2 that it is preferable to pick neighbours of black cells as `Adapt` candidates. Rather than probing them one by one to find the best candidate, we could have changed all neighbours of black cells into black, which is exactly what erosion does. It barely costs any calculation effort and in less than a second the amount of *unknowns* is reduced drastically due the introduction of many black cells. The catch however is that the result from erosion is often unappealing, even using a very small block kernel such as 2×2, which can be seen in Figure 28b.
The `specialErode` function first splits the *base puzzle* into four regions called `LT` (left-top), `RT` (right-top), `LB` (left-bottom), and `RB` (right-bottom) of approximately equal size. This is shown in Figure 26.
Each of the regions use a different shaped kernels, which are depicted in Figure 27.
The same principles apply to the regions, so only one region need to be explained. If we take for example `LT`, we want to turn the top and left neighbour of a black cell also into black cells. However for appealingness reasons, we apply conditions for when this is allowed. The left and top neighbour are treated as separate cases, so it is allowed that only one of them is turned into black. If we take the left neighbour as a candidate for it to be turned into black the following two conditions need to hold: (1) the left neighbour of the candidate

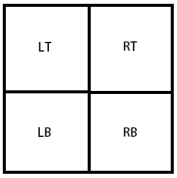

Figure 26: RegionKernels

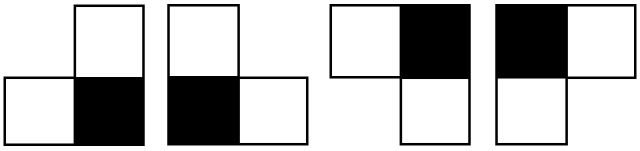

Figure 27: (a) kernel for `LT`; (b) kernel for `RT`; (c) kernel for `LB`; (d) kernel for `RB`

is white or the candidate has no left neighbour and (2) the candidate is not included in the *blacklist* set. If we take the top neighbour as candidate, then for the first condition it checks its top neighbour instead. In other words, it can also be described as the cell that is two places away from the black cell in a certain direction. This check is needed, because if turning the candidate into black would merge the two black cells, the detail that these two belong to separate features gets lost. This especially maintains the clarity of contours in the *base puzzle* that are very close towards each other.
This `specialErode` function works under the assumption that the main object depicted by Nonogram is in the centre of the image. That is the reason why each of the erode kernels expands the black cells away from the centre of the image. Figure 28c depicts the result of applying `specialErode` on Figure 28a.

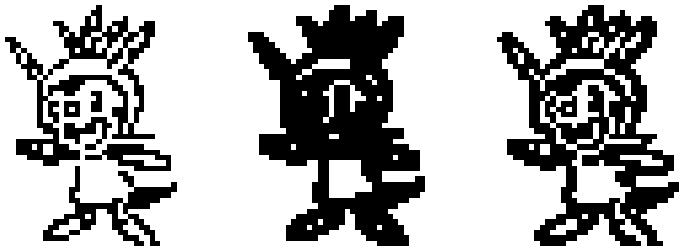

Figure 28: (a) Initial *base puzzle*; (b) Binary erosion with a block 2×2 kernel on the *base puzzle*; (c) Modified erosion on the *base puzzle*

4.3 Experiments

Section 4.3.1 examines the priority levels that are used in the `modifiedAdapt` function through tests and Section 4.3.3 describes what kind of parameter combinations are useful for the program. After acquiring these combinations, they can be established as main configurations for *ModifiedNonogramGenerator* so that the user only has to provide the input image and pass parameters such as the desired Nonogram size and the configuration.

4.3.1 modifiedAdapt Configurations

It is suggested to use all the priority tags presented in this thesis. Tests of alternative configurations where not all priority levels are applied have been run. Each configuration has been run 3 times to create simple Nonograms and the average of results is taken for Table 1.

Configuration	*High*	*Mid*	*Low*	*Blacklist*
`HMLB`	true	true	true	true
`MLB`	false	true	true	true
`HLB`	true	false	true	true
`LB`	false	false	true	true
`HML`	true	true	true	false
`ML`	false	true	true	false
`HL`	true	false	true	false
`L`	false	false	true	false

Table 1: The priority configurations

Issues of Lines:

Angled line segments of 1 pixel thick are described by clues of length 1, which especially are part of switching components. For a theoretical example, we first present a 10×10 image depicting a diagonal line of 1 pixel thick in Figure 29a where all the cells are unknowns. *Blacklisting* (with $k = 10$) serves no purpose here and can be left out in the testing, which leaves only `HML`, `ML`, `HL`, and `L`. The resulting Nonograms are depicted in Figure 29.
They show that `HML` and `ML` yield similar results and the thickened line can be attributed as an effect from including *mid priority* candidates since `HL` and L do not have it. Using only `L` generates a pattern like structure at one side of the line and `HL` generally generates a result that is closest to the original line without affecting its thickness, although the trajectory has been distorted a bit.

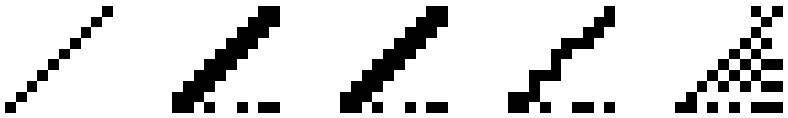

Figure 29: (a) input image; (b) result HML; (c) result ML; (d) result HL; (e) result L

To examine how the problem is translated in an actual situation, we pick an input image which has the thin, angled line structure occurring frequently as illustrated on the left of the top row in Figure 30. It is originally a binary image, which leads to a straightforward initial *base puzzle*, that also has all its cells as unknowns. Secondly, it is small enough to make it uniquely solvable within reasonable time even with the original `Adapt` algorithm to serve as a comparison.

Runtime:

If we do not make a distinction between the configurations that use blacklist or not, then from shortest runtime to longest they are ranked as follows: `HML` variants (both with blacklisting and without), `HL` variants, `ML` variants, and then `L` variants. In this specific example, `HMLB` is about 7 times faster than `LB` and `HML` about 11 times faster than `L`.
`HML` variants, however, have added the highest total amount of black cells in `modifiedAdapt` which makes it most likely that it had made the puzzle uniquely solvable through sheer numbers rather than picking candidates wisely. Also, if we compare `HML` variants to `HL` we

Figure 30: Input image Charmander of size 30×31. The left image on the top represent the original input image and the one on the right has `specialErode` applied on it. Each column represent a possible uniquely solvable Nonogram generated with the configurations described in Table 1 and are as follows from left to right: `HMLB`, `HML`, `MLB`, `ML`, `HLB`, `HL`, `LB`, `L`

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
`HMLB`	34.3	65.3	33.9	54.0	32.3	0.3	7337.7
`HML`	23.7	41.7	34.7	66.7	26.7	0	7765.7
`MLB`	82.0	65.0	32.1	0	69.7	0.3	12189.0
`ML`	81.7	60.7	32.0	0	67.3	1.0	12945.7
`HLB`	45.0	65.7	31.7	52.7	0	13.0	7720.7
`HL`	44.0	55.0	33.0	66.0	0	13.3	8739.0
`LB`	239.3	67.3	32.2	0	0	70.7	34458.0
`L`	284.3	52.0	32.5	0	0	73.0	37685.0

Table 2: Results that correspond to Figure 30

see that the number of *mid priority* cells that need to be turned black to be higher than the number of *low priority* cells, which means that *mid priority* candidates here are not ideal picks for reducing the amount of unknowns.
The white polyominos in this image that are considered blacklisted are the sparkle in the eye, the toes, jaw and the upper part of tail. In this example *blacklisting* does not impair the speed very much. This might be partially explained though because these blacklisted cells are not suitable candidates that reduce the unknown count drastically to begin with, since they are rarely picked in the other cases except the white in the eye and the toes. Further research on how the behaviour is affected when there exists a great number of blacklisted candidates is needed. From the examined cases, with the use of `ML` or `HML` variants priority levels, most Nonograms are uniquely solvable before requiring the need to add many low priority levels, which means that it is rarely the case that blacklisted candidates are required to become black in `modifiedAdapt`.

Complexity:

Blacklisting on average create Nonograms with a slightly higher complexity than without. For this input image, we also see that there is little variance in the complexity of the Nonograms that use blacklisting.

Appealingness:

From a small survey, where all the results were shown including the reference input image, it was determined that the images generated with `HLB` are considered most attractive, with the one in the bottom row in great particular. The extra speckles on the legs were not

found distracting in that image. The speckles in `L` were considered far more noticeable and for example makes the shape of the legs less clear, albeit still acceptable in the sense that it is preferred more than some issues found in other categories.
`HML` is the least appreciated category, followed by `HMLB`, due to the blottiness around the chin in `HML` which is regarded as very unappealing and sometimes the added black cells create the effect of an image getting crossed out with vertical or horizontal lines so that the image looses the clarity of its shape. The reason why the "extended lines" structure is created can be explained by the fact that in the general case long clues have less possibilities to be fixed in a line than short clues, so more black knowns can be determined. Nevertheless the relationship between row descriptions and column descriptions is very complex, because short clues can result in larger gain in knowns in some sitations: for example discovering a black cell from a chunk of short length can lead to determining more white knowns that describe where the chunk cannot appear in that line, than for a chunk of longer length.
Other minor complaints revolve around the inconsistent line thickness found in the `HML` and `ML` variants (the contours around the foot, leg, and arm are sometimes thicker than the rest of the image), the entire black right eye, the nostrils connected to the mouth, and that the left eye is not visible in `HML`.
There is no clear sign of positive nor negative reaction towards this Nonogram that is created with `specialErode`.
Relying on the non-deterministic nature of the algorithm, it is possible that small white polyominos are left white without having the involvement of blacklisting.

Conclusion:

Relying on `HL` alone is problematic when the structures described by *high priority* are not present in high number, because it reverts back to the original problem with examining *low priority* candidates. Therefore, even though the inclusion of checking *mid priority* does not yield the most appealing results, we still use the `HMLB` configuration as the default.

4.3.2 Uniquely Solvability of Base Puzzles

The authors of [3] concluded from tests that were conducted on randomly generated Nonograms of size $s \times s$, with s ranging from 10 to 50, that for small and large black percentages nearly all puzzles are uniquely solvable or in some cases "almost solvable" leaving small switching components. The results also reveal that the larger the Nonogram, the greater the amount of unknown cells it will contain. The black percentage is on average over 55% for the Nonogram to be uniquely solvable (as determined with *FourSolver*: the solver which is used to create hard puzzles).

Meaningful binary images do not represent random patterns, so these test results do not exactly provide a comment on the solvability of these, however their total amount of unknowns is probably less worse than a random Nonogram of the same dimensions. For these images to be recognisable they also usually have neither a high nor a low amount of black cells and it commonly is somewhere in the range 20 to 40 percent, which for the random generated Nonograms nearly always leads to unsolvable puzzles. While it should be a little bit better due to non-randomness, it is hard to tell how "close" they are to a uniquely solvable Nonogram: this can be for example defined as in the minimum amount of black cells that need to be added so that the initial *base puzzle* becomes uniquely solvable, but this on itself is yet a new complex problem and no attempt will be made to give an answer to this.

It is difficult to describe the nature of uniquely solvability in Nonograms. For instance the initial amount of *unknowns* in the *base puzzle* is not a good indicator of the runtime for *modifiedNonogramGenerator*, nor does it provide any information on the amount of black pixels that needs to be added in the `modifiedAdapt` loop. There has been a case where a

partial solution with 1304 unknowns (out of a total of 1350 cells) has been reduced to 12 unknowns with the addition of one black cell by `Adapt`.

There are however some main characteristics exhibited in the *base puzzles* created from the `edge3×3`, `edge9×9`, and `chooseFill` methods. The `edge3×3` method produces contours that are thin and if gradient response is dense enough to produce black clusters, they are relatively small, so clues on average are extremely short and chunks can hardly be fixed in any of their respective lines. Typical results from `chooseFill` and Laplacian9×9 have black cells that are clustered tightly to each other forming larger shapes. These shapes are described by longer clues, so that the line descriptions with those clues tend to "give away" a starting step for solving.

It is also to note that *base puzzles* from `edge3×3` also introduce far more *mid priority* candidates than *base puzzles* from edge9×9 or fill30. This is because black cells within large black clusters do not have any white neighbours and the *mid priority* candidates are located outside the perimeter of the black clusters. This means that the *base puzzles* from `edge3×3` result into having much more solver calls in each iteration of `modifiedAdapt`, so on average to add one new black cell in `modifiedAdapt` it already takes more computations.

4.3.3 ModifiedNonogramGenerator Configurations

This section describes the parameter combinations to consider which of them should be considered good configurations. All the configurations use the evaluation function described in Section 4.2.2 and all the priority levels including blacklisting are utilised. The following parameters wil be used, where `fill30AndEdges` is a parameter that has not been defined before in the thesis:

- `edge3×3`: Using Laplacian Block Kernel 3×3 and `weakEdgeReduce` to yield *base puzzle*.
- `edge9×9`: Using Laplacian Block Kernel 9×9 and `weakEdgeReduce` to yield *base puzzle*.
- `fill30`: Applying thresholding on the *base image* to yield a *base puzzle* with a black percentage of approximately 30%.
- `fill30AndEdges`: Performing both `fill30` on the greyscale *base image* and `edge3×3` on the greyscale *base image* to yield two images where the *base puzzle* is constructed as a result from performing the OR operation on the two images.

The primary goal of `specialErode` is to reduce the runtime of the Nonogram generation process. In general an image without applying erosion on it looks more appealing, therefore when erosion is not needed, it is preferred to be avoided. `specialErode` is only additionally tested with an attempt to improve results that are not satisfactorily fast enough. It is expected that *base puzzles* from `edge3×3`, compared to other methods to create *base puzzle*, take the most time to become uniquely solvable Nonograms.
The combinations described in Table 3 are considered interesting and they will be tested for a set of input images in Section 5.

Configuration	Parameters
1	`edge3×3`
2	`edge3×3` + `specialErode`
3	`edge9×9`
4	`fill30`
5	`fill30AndEdges`

Table 3: Main configurations for *ModifiedNonogramGenerator*

5 Suitable Input Images

The amount of input image possibilities is limitless and to limit it down we only show several images from the content categories described in Subsection 5.2, but before that we also define the image types in Section 5.1. All the created Nonograms in Section **??** are of the simple difficulty.

5.1 Input Image Categories

Input images are put into one of three groups that are defined as follows:

1. **Photorealistic**: photos often consist of pixel intensity distributions that occupy all the possible intensities from the 0 to 255 range. When plotting their intensity histogram, the frequency usually makes up a smooth curve.

2. **Cartoon**: Cartoons are abstract representations with flat colours and often thick contours; but not always. Nevertheless their pixels may also occupy the entire intensity range, but a small number of intensities come in far greater frequency than others, so that there are clear peaks in the intensity histogram.

3. **Pixel art**: a definition of pixel art is given in [11]. Since they usually come as images with small dimensions (pixel art was originally used to fit on low resolution screens), they do not need downscaling. Pixel art can be coloured as found in the examples in the experiments of this section or the images contain only contours as seen in Figure 30 in Section 4.

5.2 Image Content

We examine the following five image content categories, where the portraits, animal and vehicle images are taken from photos:

- Portrait of human face
- Animal: dog, cat, etc.
- Vehicle: car, boat, train, airplane, etc.
- Cartoon character
- Pixel art

Human portraits have often be used as input images in Nonogram generating programs to illustrate their capabilties, so we expect them to be suitable test cases. Animals typically have a skin or fur that forms coarse texture patterns and we want to examine how this affects the Nonogram construction problem. Photos of vehicles are in landscape format where the width of Nonogram is greater than its height. For simple cartoon style images it is easy for a human easy to perceive what the important features are since this art style emphasises them with clear contours and surfaces with flat colours, but *ModifiedNonogramGenerator* makes no assumptions on the input image. Pixel art images that consist only of contours have previously be seen in Figure 30 that they make for extremely appealing initial *base puzzles*, but pixel art with colours have not been examined yet whether *ModifiedNonogramGenerator* can turn them into recognisable and appealing Nonograms.

5.3 Characteristics for Recognisable Images

Taking into account the limitations of a small dimensions of a Nonogram and the fact that it can only portray black and white cells, it requires input images to adhere to some characteristics.

Issues when Containing Multiple Objects
Depth in a binary image cannot be expressed, so that images with overlapping objects or images depicting some object with a background behind it lead to confusing Nonograms. Therefore the object should be contained in a flat background. If images depict multiple objects or there is a main object with backdrops behind it, the objects are not allowed to touch each other. However, due to the low resolution of Nonogram image sizes, it is difficult to have the background objects be still identifiable for what they are after downscaling. For this reason, it is even recommended to submit images depicting only one object.

Lack of Background
ModifiedNonogramGenerator cannot segment main object(s) from background, and the user should do this manually and submit an image with a background that consists of flat colour. It is not only for recognisability reasons. It is also beneficial for the thresholding phase when you have a white background, because the cells that are chosen to be black are definitely features from the object. For edge detection, it makes it easier to extract the bounding contour of object(s). Therefore all the background pixels in the input images have either been manually turned entirely into white or they have been picked in such a way that the background is entirely of one intensity.

Minimum Resolution
The minimum resolution describes the image dimensions where an object is still recognisable by an average human. Not all types of objects can therefore be expressed in a Nonogram, especially if they contain a lot of complex details.

5.4 Experiments

Several images of each category have been tested and they are depicted in Figure 31 - Figure 46 together with the uniquely solvable Nonogram that is created from each configuration; the first columns depict the *base image*, the second columns the result after applying Configuration 1, the third columns the result after applying Configuration 2, and so forth.

The time that is needed to generate a uniquely solvable Nonogram from an input image is measured in seconds with `ctime` library in C++. Results are shown in Table 4 - Table 19. The speed comparison is not meant as a benchmark to test whether it is a method that creates Nonograms as fast as possible in the practical case (as it could be much quicker when ran on more powerful computers), but as an indicator of the amount of computations required to turn a *base puzzle* into a uniquely solvable Nonogram. Next to this, we have also distinguished some rating groups because users have certain time thresholds for the waiting time of what they find satisfactory or not. They are as follows:

- **Fast**: within 10 seconds
  base puzzles, before entering the `modifiedAdapt` function loop, that require very few black cells to be added can be be made uniquely solvable within a few seconds.
- **Moderately fast**: within 1 minute
- **Acceptable**: within 5 minutes (300 seconds)
- **Slow**: longer than 5 minutes

Figure 31: Portrait 1: middle aged man

Figure 32: Portrait 2: old woman with wrinkles

Figure 33: Portrait 3: contains high frequency of cells with low intensity

Figure 34: Portrait 4: contains light hair

Figure 35: Animal 1: close up of a dog's face

Figure 36: Animal 2: another dog

Figure 37: Animal 3: cat

Figure 38: Vehicle 1: car

Figure 39: Vehicle 2: boat

Figure 40: Vehicle 3: airplane

Figure 41: Vehicle 4: train

Figure 42: Cartoon 1: (Uncle Scrooge) image that consists of thick dark contours and coloured surfaces

Figure 43: Cartoon 2: (Yotsuba) image that consists of contours

Figure 44: Cartoon 3: (Hello Kitty) image that consists of flat colours

Figure 45: Pixel art 1: (Bulbasaur) contains a lot of details for a small image

Figure 46: Pixel art 2: (Larvitar) contains features that are dispersed from each other

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	65	83	37.19	23	31	0	11367
2	0	23	49.11	0	0	0	0
3	2	28	44.96	5	4	0	345
4	1	30	31.19	2	7	1	106
5	18	58	43.85	19	4	0	934

Table 4: Portrait 1: middle aged man

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	99	107	50.80	70	6	0	4261
2	0	9	63.35	0	0	0	0
3	10	83	49.20	27	2	0	908
4	179	69	33.25	17	33	0	5717
5	46	83	54.15	11	3	0	527

Table 5: Portrait 2: old woman with wrinkles

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	257	88	37.21	21	55	0	22122
2	7	33	50.95	30	1	0	871
3	1	13	51.53	0	1	0	2
4	1	21	31.32	5	2	1	36
5	5	56	47.00	8	2	0	134

Table 6: Portrait 3: contains high frequency of cells with low intensity

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	331	91	41.44	29	41	0	21087
2	0	17	55.22	0	0	0	0
3	1	38	49.44	0	0	0	0
4	1	21	31.22	3	2	0	29
5	36	86	50.39	16	4	0	1070

Table 7: Portrait 4: contains light hair

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	1023	135	40.09	38	44	0	25506
2	0	29	55.91	0	0	0	0
3	192	83	40.00	4	13	0	2694
4	664	79	37.09	6	130	2	12750
5	141	103	44.95	26	7	0	1847

Table 8: Animal 1: close up of a dog's face

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	292	92	39.52	14	90	0	41339
2	5	22	51.48	27	2	0	781
3	1	40	43.33	0	2	0	6
4	5	37	31.19	5	3	0	265
5	20	40	44.57	11	2	0	746

Table 9: Animal 2: another dog

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	345	118	31.37	19	98	0	44063
2	19	79	42.00	15	3	0	489
3	2	49	44.51	0	3	0	25
4	29	43	33.03	10	9	0	2319
5	35	45	42.40	13	9	1	2663

Table 10: Animal 3: cat

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	185	122	35.39	20	8	0	3231
2	1	18	48.69	0	0	0	0
3	5	34	42.75	3	2	1	66
4	0	14	30.72	2	0	0	3
5	34	52	45.07	11	3	0	725

Table 11: Vehicle 1: car

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	34	56	22.46	18	14	0	5224
2	1	25	32.63	0	0	0	0
3	10	37	22.23	6	6	0	1819
4	0	7	30.51	0	0	0	0
5	0	19	34.23	0	0	0	0

Table 12: Vehicle 2: boat

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	58	59	24.18	17	59	0	15655
2	3	24	32.79	14	1	0	362
3	0	22	48.73	0	0	0	0
4	18	29	32.67	3	34	0	4421
5	24	38	41.27	9	11	0	2207

Table 13: Vehicle 3: airplane

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	38	40	38.06	16	15	2	5647
2	0	16	50.73	0	1	0	2
3	0	11	51.09	0	0	0	0
4	0	16	30.85	0	1	0	2
5	1	24	46.91	2	1	2	8

Table 14: Vehicle 4: train

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	114	69	33.05	27	18	1	8961
2	9	34	46.77	31	1	0	907
3	4	30	44.55	4	1	1	159
4	0	18	31.14	2	2	1	12
5	17	56	47.73	20	3	1	541

Table 15: Cartoon 1: (Uncle Scrooge) image that consists of thick dark contours and coloured surfaces

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	2233	84	42.67	48	183	1	46013
2	17	62	46.17	12	1	0	402
3	204	75	41.17	24	38	1	14780
4	169	58	33.56	9	38	1	4809
5	437	69	43.17	21	63	1	12730

Table 16: Cartoon 2: (Yotsuba) image that consists of contours

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	371	111	26.45	19	36	0	10806
2	23	32	39.15	19	3	0	1356
3	0	16	48.8	1	3	0	8
4	542	39	39.05	35	291	1	64987
5	238	93	33.3	27	39	0	14201

Table 17: Cartoon 3: (Hello Kitty) image that consists of flat colours

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	152	98	41.27	50	22	0	6711
2	1	13	56.95	0	0	0	0
3	17	41	50.53	18	11	0	1057
4	0	9	32.44	4	2	2	36
5	11	31	55.97	25	4	0	569

Table 18: Pixel art 1: (Bulbasaur) contains a lot of details for a small image

Configuration	Time (s)	Complexity	Black (%)	High	Mid	Low	Solver calls
1	76	83	27.28	64	46	0	15748
2	0	37	34.34	3	0	0	105
3	14	60	28.23	25	7	0	1823
4	9	35	31.72	20	4	0	641
5	11	40	31.92	20	4	0	669

Table 19: Pixel art 2: (Larvitar) contains features that are dispersed from each other

5.5 Evaluation of the Configurations

Results do not single out a configuration that works best for an arbitrary input image. However we do see reoccurring characteristics from the five configurations and they are described below.

Configuration 1:
The drawback of this configuration is that when even the *base puzzle* has high appealingness, it is not always guaranteed in the actual uniquely solvable Nonogram when it requires the need of adding a lot of pixels in the `modifiedAdapt` function. The runtime is also unpredictable as it can take more than half an hour for some input images to generate a uniquely solvabe Nonogram while it works acceptably fast for others. On average, the waiting time is considered to be extremely slow which makes this configuration impractical.

Configuration 2:
The uniquely solvable Nonogram is not particularly appealing, but the average runtime is extremely fast.

Configuration 3:
Sometimes the most recognisable Nonogram is created with this configuration; as seen in Figure 42 and Figure 43 of the cartoon category with thick contours, this Configuration performs best at extracting the contours. The average runtime is acceptably fast.

Configuration 4:
It is capable of creating appealing Nonograms, however it does not perform well when low intensity pixels in *base image* are in abundance so that it excludes a lot of important details. The runtime is moderately fast, but can also end up slow in the cases of Figure 32 and Figure 35. These are images that produce *base puzzles* with a lot of dispersed black cells and less of tightly clustered, large black surfaces.

Configuration 5:
In most cases, it has the most appealing Nonograms, however since the initital *base puzzles* from both Configuration 1 and 4 directly affect this result, it might not end up appealing when either one of them did not create a *base puzzle* of good quality.

6 Conclusion and Future Work

We have given an automatised approach for constructing recognisable Nonograms called *ModifiedNonogramGenerator*, which has improvements over the original *BasicNonogram-Generator* which includes: a method to turn a *base puzzle* into a uniquely solvable Nonogram with less computation (to result in a slightly faster algorithm), removes the user requirement of setting parameter values themselves and *ModifiedNonogramGenerator* is more robust in creating recognisable and appealing Nonograms where *BasicNonogramGenerator* might fail to capture important features from an input image.

In this section we will mention some of the findings regarding several proposed improvements of the Nonogram generating process.

Edge detection is better than thresholding at extracting all the important features from object, however both Laplacian 3×3 and Laplacian 9×9 with `reduceWeakEdges` have the issues of oversensitivity and detecting features a human viewer would not consider important. These are for examples the wrinkles on a face. In the case of Laplacian 9×9 there is little freedom for the *base puzzle* as it has quite a strong sensitivity in the gradient maps as seen from the pixels with low intensity, however for Laplacian 3×3 the pixels in the gradient map are varied which leaves far more freedom to decide which should be edges. That the `lowerEdgeThreshold` is set as high as 200 and the `upperEdgeThreshold` is set to 255 in `reduceWeakEdges` is to make sure that even extremely weak gradient pixels were included in the *base image* to raise up the black percentage count. However, the 30 to 40 percent rule for black percentage is probably meant to be a recommendation when applying thresholding on a non-edge detected *base image*.

It turns out that Laplacian 3×3 is not very useful by itself, except when the user wants a Nonogram with a high complexity. Due to the extreme long waits Configuration 1 is quite unreliable. `modifiedAdapt`, while faster than `Adapt`, is still slow. `specialErode` works well at reducing runtime, but the result is not appealing at all. Using thresholding and Laplacian 3×3 together can overcome each other's shortcomings, for example the features that were unable to be caught with thresholding are often expressed through edge contours and the extreme long wait in `Adapt` are alleviated when it includes some large black surfaces that were captured from thresholding. In that case, for future work the parameter settings for `reduceWeakEdges` should be optimised so that it works well in combination with thresholding. Setting the `lowerEdgeThreshold` on a lower value creates a *base puzzle* with far less abundant features and it is definitely possible to filter out the creases and wrinkles on a face, since these have a weak gradient.

There is room for speed-up. Only one pixel is chosen to be changed into black in the `modifiedAdapt` function, but the algorithm would probably go faster if larger steps are taken, a changing several pixels at once.

Another still remaining challenge in this problem is to provide heuristics for recognisability and appealingness. If it is possible, then different kinds of algorithms could have been applied to the Nonogram construction problem, such as Natural Computing Algorithms (which is the superset of Evolutionary Algorithms and Swarm Based Optimisation Algorithms) that rely on heuristics and are recognised to be succesful at finding good solutions in a large search space.

References

[1] Website griddlers [accessed 28.8.2013]. `http://www.griddlers.net`, 2013.

[2] K. J. Batenburg and W. A. Kosters. A discrete tomography approach to Japanese puzzles. *Proc. Belgian-Dutch Conf. Artificial Intelligence (BNAIC 2004)*, pages 243–250, 2004.

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

[4] K.J. Batenburg, S.J. Henstra, W.A. Kosters, and W.J. Palenstijn. Constructing simple nonograms of varying difficulty. 2009.

[5] K.J. Batenburg and W.A. Kosters. Nonograms. 2012.

[6] R.A. Bosch. Painting by numbers. *Optima*, pages 16–17, 2001.

[7] A. Bovik. The essential guide to image processing. 2009.

[8] J. Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 679–698, 1986.

[9] E.R. Davies. *Computer and Machine Vision: Theory, Algorithms, Practicalities*. Elsevier, 4th edition, 2007.

[10] S.J. Henstra. From image to nonogram: Construction, quality and switching graphs. *Master Thesis (Universiteit Leiden)*, 2011.

[11] T.C. Inglis and C.S. Kaplan. Pixelating vector line art. *Proceeding NPAR '12 Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, pages 21–28, 2012.

[12] O.F. Lopez. A compressive sensing approach to solving nonograms. *Master Thesis (University of Texas at Austin)*, 2013.

[13] E.G. Ortiz-Garca, S. Salcedo-Sanz, J.M. Leiva-Murillo, A.M. Prez-Bellido, and J.A. Portilla-Figueras. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics*, pages 750–760, 2007.

[14] J.T. Tsai. Solving Japanese nonograms by Taguchi-based genetic algorithm. *Applied Intelligence*, pages 405–419, 2012.

[15] N. Ueda and T. Nagao. Np-completeness results for nonogram via parsimonious reductions. 1996.

[16] W.A. Wiggers. A comparison of a genetic algorithm and a depth first search algorithm applied to japanese nonograms. *Twente Student Conference on IT*, 2004.

[17] C.H. Yu, H.L Lee, and L.H. Chen. An efficient algorithm for solving nonograms. *Appl. Intell.*, pages 18–31, 2011.