Utrecht University

The concept and automatic generation of the Curved Nonogram puzzle

by Tim de Jong

Master Thesis
ICA-4123689

Game and Media Technology
Utrecht University

supervised by
Marc van Kreveld
and Maarten Löffler

July 8, 2016

Abstract

In this paper we introduce a variation on the Nonogram puzzle that we call the Curved Nonogram. The rules for colouring are similar, but instead of being constrained to a grid it is structured by arbitrary curves that can take any shape. This allows for puzzles that feel more unique and can have more beautiful images hidden in them.

Together with the introduction of this new puzzle, we define classes of solvability for it and provide measurements to judge its aesthetics. We also propose an algorithm that can generate these Curved Nonograms, taking the desired solution image as input. We test this algorithm on 16 varied input images and find that all of the outputs have a unique solution - an important requirement for puzzle-book puzzles, and at least 10 of them satisfy the aesthetic criteria sufficiently that they could be used in a puzzle book.

1 Introduction

1.1 Nonograms

Nonograms, also known as Paint-by-numbers puzzles or Japanese puzzles, are a popular type of pencil-and-paper puzzle that can be found in puzzle books around the globe. The goal of the puzzle is to make a picture appear by colouring the correct cells in a square grid, according to the given row- and column descriptions. Figure 1 shows an example of a Nonogram. The description of a row or column consists of a sequence of numbers, which are the lengths of the sequences of coloured cells that appear in it. For example, the number 3 on the second row means that this row will have one uninterrupted sequence of three coloured cells. The puzzler can use this information to reason which cells should be coloured and which should be left empty.

A Nonogram starts as an empty square grid with descriptions, and ends with an image revealed in the grid. The discovery of the image acts a reward for solving the puzzle.

	2	1	4	1	2
1					
3					
1 1 1					
1 1 1					

(a) Empty

	2	1	4	1	2
1			■		
3		■	■	■	
1 1 1	■		■		■
1 1 1	■		■		■

(b) Solved

Figure 1: An example of a Nonogram. The puzzler uses the descriptions at each row and column to fill cells in the empty puzzle (a) in order to find the image formed by the solved puzzle (b).

1.2 Curved Nonograms

In this paper we introduce the Curved Nonogram, a new variant on the classic Nonogram puzzle. A Curved Nonogram consists of a set of arbitrary curves enclosed by a border. An example is shown in Figure 2. The only constraints we impose on the curves are that (1) they lie completely within the border; (2) the endpoints of each curve lie on the border; and (3)

that no more than two curves may intersect at the same point.

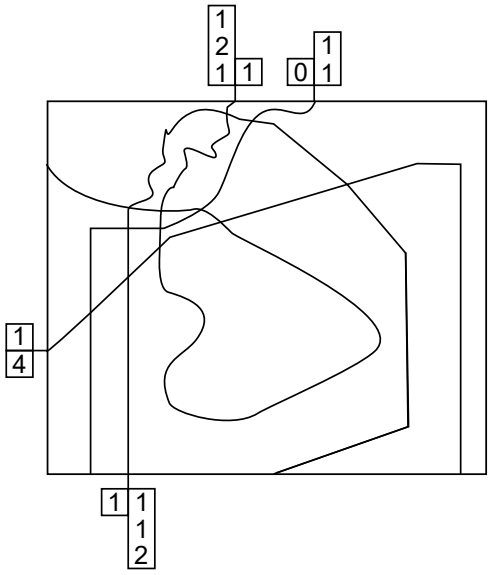
Instead of row- and column descriptions, the Curved Nonogram has a description for both sides of each curve. In the same way that the descriptions of a classic Nonogram give information about the sequence of cells that form a row or column, this description gives information about the sequence of faces that are incident to that side of the curve. For example, see the description "1 1" in Figure 2. Following this side of the curve from start to end, you will encounter one coloured face, one or more empty faces, and one coloured face again, in that order. When reading a description, one should start with the outermost number and end with the innermost number. For example, in Figure 2, the description on the right side of the curve that starts at the bottom of the puzzle should be read as "2 1 1".

The purpose of the three constraints given above is to make the Curved Nonogram clear and easy to grasp for puzzlers, especially if they already have experience with classic Nonograms. Having a single border makes it clear which faces are part of the puzzle. We require the endpoints of the curves to lie on the border to make it a clear start- and endpoint when tracing the faces incident to the side of a curve. The reason for not allowing three or more curves to intersect in a point is also a practical one. At such a multi-intersection, some curves and faces are only incident to each other in a single point. Whether this form of adjacency counts or not is not clear for the puzzler, so it would have to be explicitly defined in the rules of the puzzle book. Because we want Curved Nonograms to be understandable without requiring the memorization of such specific rules, constraint (3) was added to prevent this situation from occurring.

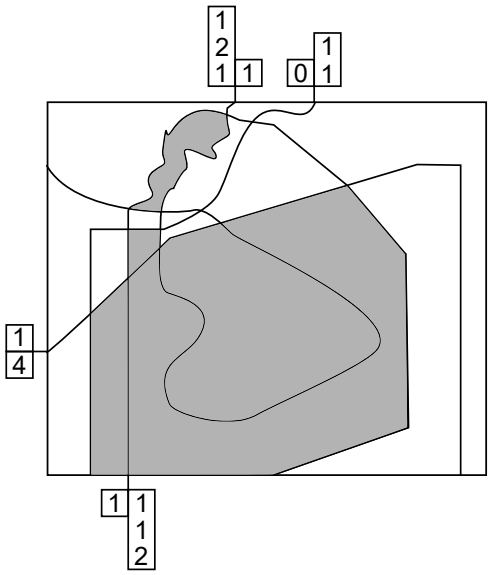
While the rules of the Curved Nonogram are similar to those of the classic Nonogram, there are several things that distinguish the Curved Nonogram from the classic version.

First of all, because the classic Nonogram consists of a square grid, the image formed by the solution is necessarily very pixellated and abstract. A Curved Nonogram is not limited by such a grid, and the image that is formed by the solved puzzle can take nearly any shape. This means that the image can more faithfully represent the image that the puzzle designer has envisioned.

Second of all, the puzzler's curiosity is triggered as soon as they see the puzzle and try to visualise the shape of the solution image hidden between the



(a) Empty



(b) Solved

Figure 2: An example of a Curved Nonogram. The descriptions are associated with one side of a curve, instead of a row or column. Again, the goal is to start with the empty puzzle (a) and find the image formed by the solution (b), which shows a house in perspective with smoke coming from the chimney.

curves. This in contrast to a classical Nonogram, where the only moment of surprise comes at the end, when the puzzle has been solved and the image has become clear.

Thirdly, Curved Nonograms offer new challenges to puzzlers, even if they are already familiar with classic Nonograms. The one most obvious is that they will carefully have to trace the path of each curve through the puzzle to see which faces are incident to it. Less obvious but more impactful is the information that can be obtained when one side of a curve comes across the same face more than once. Such an occurrence gives the puzzler a new type of clue that is not present in classic Nonograms. While this aspect may be too subtle or confusing for the beginning puzzler, it could be introduced over time in a puzzle book, with only the hardest puzzles really requiring this information in order to progress. We choose to allow this situation in our rules, because we think that its occurrence is one of the factors that make Curved Nonograms stand out from classic Nonograms - especially for the more experienced puzzler. Another interesting difference is that a cell in a classic Nonogram is referred to in the descriptions of exactly one row and one column. A face in a Curved Nonogram, on the other hand, can be referred to by many more descriptions, namely one for each curve that bounds it.

1.3 Automated generation of Curved Nonograms

In addition to the concept of Curved Nonograms, we will also introduce an algorithm that generates a Curved Nonogram from a given image in vector graphics format. This may prove useful for a puzzle designer, who will only have to draw his envisioned solution image and let the algorithm do the rest. Our intent is also to study the output of this algorithm, in order to get a better understanding of Curved Nonograms. We would like to generate Curved Nonograms that satisfy the following conditions:

1. The generated puzzle should adhere to the three rules of Curved Nonograms described above
2. The generated puzzle should be visually unambiguous
3. The generated puzzle should be aesthetically pleasing
4. The generated puzzle should have a unique solution
5. The solution image of the generated puzzle should look like the input image
6. The outlines of the solution image should be sufficiently obscured in the unsolved puzzle

The first condition is required to generate a valid Curved Nonogram.

The second condition concerns potential sources of visual ambiguity such as small faces, intersections at a small angle, and narrow parts of faces (i.e. where two curves almost touch). The generated puzzle should be solvable by a human puzzler, which makes it important that the structure of the puzzle is clear to the naked eye.

The third condition concerns the aesthetics aside from ambiguity, and is perhaps the hardest to define mathematically or be solved by a computer. We still choose to add it to the list, because this condition would certainly be a goal of a human puzzle designer; consequently, an algorithm that automates this part of the designer's job should be judged by it. Examples of this criterion include making fluid curves, and not making large empty faces.

The fourth condition requires the puzzle to have a unique solution. We will show later that not all (Curved) Nonograms have a solution, and that it is also possible that a puzzle has multiple solutions, none of which is desirable.

The fifth condition requires the generated puzzle to have a solution image that looks like the input image. The weight that is given to this condition determines the level of control that the user has over the output.

The sixth and last condition states that the outlines of the solution image should be well hidden in the unsolved puzzle. The risk that the puzzler can guess the image before they have solved the puzzle is inherent to Curved Nonograms. This risk should be mitigated as much as possible in the generated puzzles.

The creation of an algorithm that satisfies all of the conditions is a difficult task that exceeds the scope of this paper. We choose to focus on the aesthetic side of the puzzle generation, because this side is the most new and unique aspect of Curved Nonograms. We develop an algorithm that generates Curved Nonograms that satisfy criteria (1), (2), (3) and (6). We then experiment with the evaluation function of the algorithm to learn which terms give the best results according to these criteria. We also measure the percentage of the generated puzzles that have a unique solution as per criterion (4), and judge how well the solution images are obscured as per criterion (5). While we do not optimise for these last two criteria, the observations may prove useful to indicate if these criteria require more focus in possible future work.

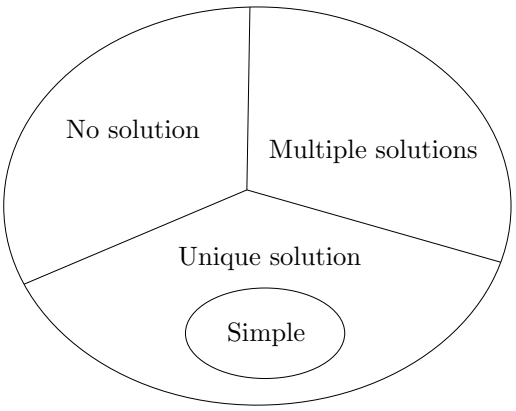


Figure 3: Relation between the four classes of solvability for (Curved) Nonograms

2 Related work

Nonogram solving Most of the literature on Nonograms focuses on solving them as efficiently as possible. Because the problem of solving a Nonogram is NP-complete [1], there is no straightforward solution to this problem. Many different types of algorithm have been applied to solving them, based on techniques from Discrete Tomography [2], chronological backtracking [3] and many others [4][5].

There are also many websites dedicated to solving Nonograms. Our algorithm uses an adapted version of the well-documented classic Nonogram solver of *webpbn.com* [6] to determine the solvability of Curved Nonograms.

Classification of Nonograms There is also research on the classification of Nonograms [4][7]. From this work we can learn that a Nonogram always falls into one of four classes of *solvability*: unsolvable, multiple solutions, unique solution, and simple. Figure 3 shows how these classes relate. A Nonogram is called *simple* if it can be solved by working on one row or column at the time. Most Nonograms found in puzzle books are simple, and this subset of Nonograms is solvable in polynomial time. Because there are no branching decisions while solving a simple Nonogram, they always have a unique solution. We will show that these same classes apply to Curved Nonograms. Of course, the Curved Nonograms generated by our algorithm should ideally always have a unique solution. While we do not implement this feature, we do determine the class of each generated puzzle in order to analyse how the generated puzzles are distributed among the classes.

Puzzle generation Perhaps due to the popularity of puzzles and puzzle games on mobile devices, a lot of recent research has been done on the automatic generation of puzzles. A Nonogram-related example is [7], which is about generating simple Nonograms of a given difficulty - roughly defined as the number of steps required to solve it. The author finds that changing the solution colour of a single face can have a large effect on the difficulty of the puzzle. Other Nonogram-generation related work explores different questions, such as how to generate them from digital RGB images [8].

There are, of course, many studies of puzzle generation for other puzzles than Nonograms. Examples are the generation of Sudokus [9], Crossword puzzles [10], and Connect-The-Dots puzzles [11]. The Connect-The-Dots paper is especially relevant, because it also deals with ambiguities in a generated puzzle. However, the specific ambiguities involved are quite different from those in Curved Nonograms, because the puzzle elements are dots instead of curves, faces and intersections.

Automated drawing aesthetics One of the things that we are trying to accomplish in this paper is to generate clear and aesthetically pleasing Curved Nonograms. The aesthetics of automated drawing are much studied in the field of Graph drawing [12], where the input is a graph, which needs to be embedded in the plane. Many studies have been done to find a set of aesthetic criteria for such an embedding [13][14][15]. Other papers offer algorithms that can be used to generate Graph drawings according to these criteria. An example of an algorithm that uses force-directed placement can be found in [13].

3 Definitions

We will start by giving our definition of the classical Nonogram in Section 3.1. This is followed by the definition of the Curved Nonogram in Section 3.2, which uses some of the same elements. Section 3.3 then defines the classes of solvability for both types of Nonogram. Finally, three measurements to judge the aesthetic of a Curved Nonogram are introduced in Section 3.4.

3.1 Nonogram

A *Nonogram* consists of a grid of square cells, which we will also refer to as the faces, and a *description* for each row and column of the grid. See Figure 1a.

A *colouring* C of the Nonogram is an assignment of colours from the set $\Sigma = \{b, w\}$ (black and white) to its faces.

Each row and column has a *sequence* of faces in it. For brevity, we will refer to this as simply the *sequence* of a row or column. For a given colouring C, each of these sequences is associated with a *colour string* over the alphabet Σ, which corresponds to the colours assigned to the faces in the sequence under C. For a colour string s of a sequence with l faces:

$$s \in \Sigma^l. \tag{1}$$

Note that a Nonogram with an $n \times m$ grid has $n + m$ sequences. The descriptions of the Nonogram are each associated with the sequence in the corresponding row or column. A description is a string of positive integers:

$$d = c_1 c_2 \dots c_k. \tag{2}$$

A description may also be an empty string, which is displayed in the puzzle as a zero. For a given colouring, the colour string s of a sequence *adheres* to the description d if it satisfies the following regular expression:

$$s \in w^* b^{c_1} w^+ b^{c_2} w^+ \dots b^{c_k} w^*. \tag{3}$$

For readers unfamiliar with regular expressions: w^* denotes a string of zero or more w's; w^+ denotes a string of one or more w's; and b^c denotes the string consisting of c times the b character. In other words, the pattern describes a sequence that contains k segments of consecutive black coloured cells, of which the lengths and order are given by the integers in the description. These segments are separated by at least one white cell, and the sequence may start and end with a segment of white cells.

A colouring C is called a *solution* of the Nonogram if each of the colour strings adheres to the description associated with its sequence. In Section 3.3, we show that a Nonogram may have no solution, a single solution, or multiple solutions.

3.2 Curved Nonogram

A *Curved Nonogram* consists of a two-dimensional arrangement induced by curves, together with a collection of descriptions. See Figure 2a for an example.

One curve is called the *puzzle enclosure*, which is a closed curve that encloses the other curves. The second type of curve are the *puzzle curves*, defined as the collection $P = \{p_1, p_2, \dots, p_n\}$. These curves are associated with the descriptions, and their start- and endpoints lie on the puzzle enclosure. The third and final type of curve are the background curves, defined as $B = \{b_1, b_2, \dots, b_m\}$. These curves are not associated with any descriptions, and their only restriction is that they are contained in the puzzle enclosure.

Like with a classical Nonogram, a colouring C of a Curved Nonogram is an assignment of colours from the set $\Sigma = \{b, w\}$ to its faces, which are the faces of the arrangement.

The two sides of a puzzle curve are each incident to a different sequence of faces. For this reason, we conceptually split each puzzle curve into two half-curves, each representing one side of the curve. A half-curve is incident to one sequence of faces, which we call the sequence of the half-curve. Thus, a Curved Nonogram with n puzzle curves has $2n$ sequences.

The rest of the definition is analogous to that of the classic Nonogram. For a given colouring, the sequences are associated with colour strings as defined in (1). Each sequence is associated with a description as defined in (2). A sequence adheres to a description if it matches the regular expression in (3). Finally, a colouring C of a Curved Nonogram is called a *solution* if each of the colour strings adheres to the description associated with its sequence.

3.3 Solvability

The classification of classic Nonograms based on solvability was discussed in Section 2, including the diagram in Figure 3. We will give a more precise definition of the solvability classes here, and extend them to the concept of a Curved Nonogram. The definitions of solvability will be given for both types of Nonogram at once, using the word Nonogram to refer to both.

If all colour strings of a colouring C adhere to their corresponding descriptions, C is called a *solution* of the Nonogram. It is possible that there are multiple different colourings of the Nonogram for which this is true. In that case, the Nonogram has multiple solutions.

It is also possible that no colouring C exists for which all colour strings adhere to their descriptions. In this case, the Nonogram has no solution.

Finally, if exactly one colouring C exists that is a solution, the Nonogram is said to have a unique solution. All Nonograms that appear in puzzle books belong to this class, because they are intended to be

solved and the solution should look like a specific picture, which can usually be looked up at the end of the book.

For a Nonogram with a solution C, we define the corresponding *solution image* as the geometric region formed by the union of the black-coloured faces in C. If a Nonogram has multiple solutions it also has multiple solution images: one for each of its solutions.

Simple Nonograms In [7] the authors define a fourth class of classic Nonograms called *simple*. These Nonograms are a subset of the Nonograms that have a unique solution. While the complete definition of a simple Nonogram can be found in the original paper, we provide a succinct definition of the used terms adapted to our definition of the classical Nonogram and extend its meaning to Curved Nonograms.

Let there be a *Fix* operation that operates on a sequence. This operation assigns a colour to those faces in the sequence that must have a specific colour, based on the description and the faces for which the colour has already been assigned. According to the definition in [7], a classic Nonogram is called simple if it can be solved by the following algorithm:

1. apply the *Fix* operation to the sequence of each row
2. apply the *Fix* operation to the sequence of each column
3. if the Nonogram is not solved, go to step 1

There are several interesting aspects to simple Nonograms. Firstly, solving a simple Nonogram only takes polynomial time. The *Fix* operation can be executed in polynomial time with the dynamic algorithm given in [4]. For a simple Nonogram, the algorithm given above must determine the colour of at least one new face in each iteration, or it would be stuck. This means that a simple classic Nonogram with n rows and m columns is solved by this algorithm in at most mn iterations.

Secondly, the given algorithm is similar to the strategy that most humans use to solve a Nonogram. This makes simple Nonograms relatively easy to solve by hand. Because of this property, most Nonograms found in puzzle books are simple.

Simple Curved Nonograms We give our definition of a simple Curved Nonogram, and show that it is also solvable in polynomial time. Re-using the *Fix* operation defined in the last paragraph, we define a Curved Nonogram to be simple if it can be solved by the following algorithm:

1. apply the *Fix* operation to each of the sequences of the Nonogram
2. if the Nonogram is not solved, go to step 1

For a classic Nonogram, this definition is equivalent to the one in the previous paragraph. In order for a Curved Nonogram to be solved by this algorithm without getting stuck, the colour of at least one new face must be determined in each loop. This means that it can take at most $2n$ loops to finish for a simple Curved Nonogram with n puzzle curves, i.e. one loop per half-curve. Because we already know that the *Fix* operation can be performed in polynomial time, we can conclude that the entire algorithm takes polynomial time as well.

3.4 Aesthetics measurements

In Section 1.3 we gave six conditions that make for a good Curved Nonogram. Two of these conditions are related to the aesthetics of a Curved Nonogram: that it should be visually unambiguous and that it should be aesthetically pleasing. In this section, we derive several concrete measurements that can be used to determine if a Curved Nonogram satisfies these two conditions. The concrete measurements in this section were derived from a list of informal criteria. These criteria describe the properties that an unambiguous and aesthetically pleasing Curved Nonogram should or should not have, and is based on a combination of intuition and explorative research. The list of informal aesthetic criteria that we use is as follows. An unambiguous and aesthetically pleasing puzzle should:

1. not have large, open areas where not much happens
2. not have convoluted, confusing areas where a lot of curves and crossings lie close together
3. not have faces with very narrow parts
4. not have curves that intersect at a very small angle

Each of these informal criteria is represented by at least one of the concrete measurements defined below.

(a) Minimum vertex distance The first concrete measurement is the minimum distance between any two vertices in the arrangement of the Curved Nonogram. If this measurement has a high value, there are no ambiguous areas with close-together vertices. Thus, this measurement can be used to assess aesthetic criterion 2.

Let $d(v, w)$ be the Euclidean distance between vertices v and w in the plane. The minimum vertex distance (m_a) is defined as follows:

$$m_a = \min_{v,w \in V | v \neq w} d(v, w),$$

where V is the set that contains the vertices of the puzzle arrangement.

(b) Largest inscribed circle of any face The second measurement is based on the inscribed circles of the faces of the arrangement. The inscribed circle of a face is the largest circle that fits in the interior of the face. If we take the inscribed circle of all faces of the arrangement, this measurement is equal to the area of the largest circle. A high value of this measurement means that the Curved Nonogram has at least one face with a large, boring open area in it, which makes it less interesting from an aesthetic perspective. Thus it is an ideal measurement for criterion 1. We define the largest inscribed circle (m_b) as follows:

$$m_b = \max_{f \in F} A(I(f)),$$

where F is the set that contains the faces of the puzzle arrangement, $I(f)$ is the inscribed circle of face f, and $A(c)$ is the area of circle c.

(c) Local geometric dilation The third and final measurement is based on the concept of geometric dilation. For this measurement, we interpret the arrangement of the puzzle as a graph G embedded in the plane. Let p and q be any two points on the edges of G, with $p \neq q$. Let $d(p, q)$ again denote the Euclidean distance between p and q in the plane. Finally, let $d_G(p, q)$ be total arc length of the shortest path in G between p and q. The geometric dilation between p and q is then defined as follows:

$$\delta(p, q) = \frac{d_G(p, q)}{d(p, q)}.$$

Geometric dilation can also be measured for the entire graph. The geometric dilation of G is defined as

$$\delta(G) = \sup_{p,q \in G} \delta(p, q).$$

A low geometric dilation can be considered a positive aspect for a puzzle. One reason is that geometric dilation functions as a lower bound for the intersection angles of the curves of the puzzle, which can be used to check criterion 4. To be precise, all angles α in the puzzle are bound by the following formula:

$$\alpha \geq 2 \sin^{-1}(\frac{1}{\delta(G)}).$$

Because we have not placed any restrictions on the shape of the curves that form a Curved Nonogram, they can in principle contain many sharp angles and other small-scale irregularities that can make the puzzle ambiguous. A low geometric dilation also limits how extreme these irregularities can be. Finally, the geometric dilation is the only measurement of the three that relates to the narrowness of faces, which relates to criterion 3. If the arrangement has a face that is at one part very narrow, or in other words, there are two curves that come very close at some point without touching, this causes the geometric dilation of the arrangement to be high.

While geometric dilation is a useful measurement, it should be noted that a lower value is not always better. An arrangement with minimum geometric dilation has very regular faces that start to approximate circles. What we really want is an arrangement with angles that are not too small, and faces that are not so narrow that they become ambiguous. Therefore, we introduce the concept of *local geometric dilation* (m_c), an alternative form of geometric dilation that is only affected by points that are within a certain distance of one another.

$$m_c = \sup_{p,q \in G | d(p,q) \leq d_{dil}} \frac{d_G(p, q)}{d(p, q)},$$

where d_{dil} is a constant that determines how close a pair of points should be to contribute to the measurement.

4 Algorithm

This section describes the algorithm that we use to generate Curved Nonograms. The algorithm takes a coloured line drawing as input, and generates a Curved Nonogram that has a solution image that is the same as the input. Because this is the first puzzle generation algorithm for Curved Nonograms, the goal is to generate Curved Nonograms that are not ambiguous and could thus be used in a puzzle book. Thus we do not focus on low computation times or aesthetic aspects that are unrelated to the ambiguity. The solvability of the puzzle is also not taken into account. However, because the puzzles are created from a solution image (the input), they are guaranteed to have at least one solution.

First, the input for the algorithm is detailed in Section 4.1. Following that, Section 4.2 describes the

structure of the solutions that our algorithm produces. A parameterisation of the solution space is given in Section 4.3. Next, the implementation details are given that are relevant to this description of the algorithm, in Section 4.4. The evaluation function to judge solutions with is then defined in Section 4.5. Finally, in Section 4.6 we give the pseudo code of the optimization algorithm that uses the aforementioned elements to find a good solution.

4.1 Input

The input to the algorithm proposed in this section is a planar arrangement induced by a set of *input curves* and a rectangle R. The faces of this arrangement are coloured either black or white.

An input curve is a chain of Bézier curves, which can be either first, second or third order. These types of chains are widely used in the SVG format, a vector-graphics format, which has the advantage that any SVG drawing tool can be used as a graphical interface to provide input for the algorithm. Another reason to use these Bézier curves is that they are not very complex.

The input curves are enclosed by R. This rectangle will be used as the puzzle enclosure of the output puzzle. While the puzzle enclosure of a Curved Nonogram can take any shape (as long as it is closed), we limit the shape to a rectangle in the input of the algorithm. This simplifies some later steps in the algorithm, like point-inclusion checks and the global optimization. An example of the input is shown in Figure 4.

We use the SVG editor Inkscape to create the SVG files from which our program reads the input. While other formats could be used to store the input, this format offers the advantage of the availability of freely available graphical editing tools. Of course, the input could be provided in text format or any other format, as long as the interpretation is the same.

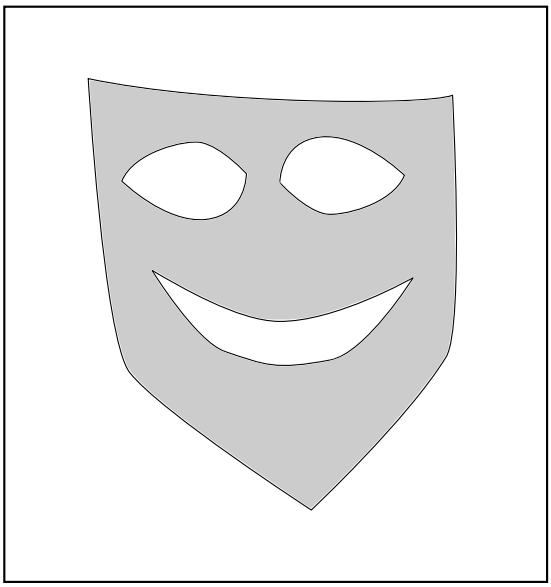


Figure 4: An example of the input of the algorithm with four (closed) input curves and one black face.

4.2 Solution space

In principle, the only restriction placed on the output of the algorithm is that it is a valid Curved Nonogram with puzzle enclosure R. This is a very complex solution space, with an unbounded dimensionality caused by the unbounded number of curves of unbounded complexity that any solution may contain. We choose to limit the solution space for our algorithm by adding additional constraints on the structure of the output.

Firstly, we let the input curves be preserved in the output. This means that we can keep the black and white coloured areas of the output exactly the same as those in the input. This allows the algorithm to perfectly achieve the previously stated goal that the solution image of the output should be similar to the input.

To be able to preserve the input curves in the output, those curves must be connected to the puzzle enclosure at both ends, as per the definition of a Curved Nonogram. The input curves are chains of Bézier curves, some of which may be closed. We choose to cut the input curves, at each point where two consecutive Bézier curves in the chain are not C^1-continuous [16]. An example of this cutting is shown in Figure 5. This procedure cuts open most of the closed chains, allowing them to be connected to the puzzle enclosure. It also serves to make the chains less long, which we expect to lead to a better puzzle. We call these new, smaller Bézier chains the puzzle curves, because each of them will become a puzzle curve in the output Curved Nonogram.

Note that closed input curves that are C^1-continuous will not be cut open by this method. We do not connect these curves to the puzzle enclosure, so they will become background curves in the output.

The second constraint describes how each of the puzzle curves is connected to the puzzle enclosure. A puzzle curve may have zero, one or two ends that already lie on the puzzle enclosure. We call the ends that do not lie on the puzzle enclosure the *extendable*

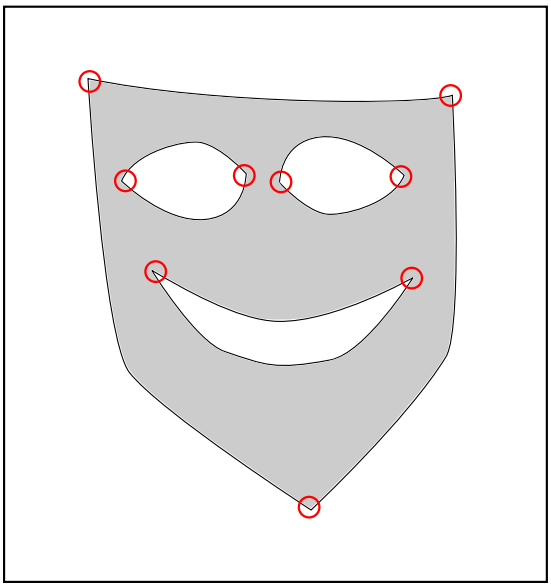


Figure 5: An example of how the input curves are cut. The four input curves are cut into nine puzzle curves.

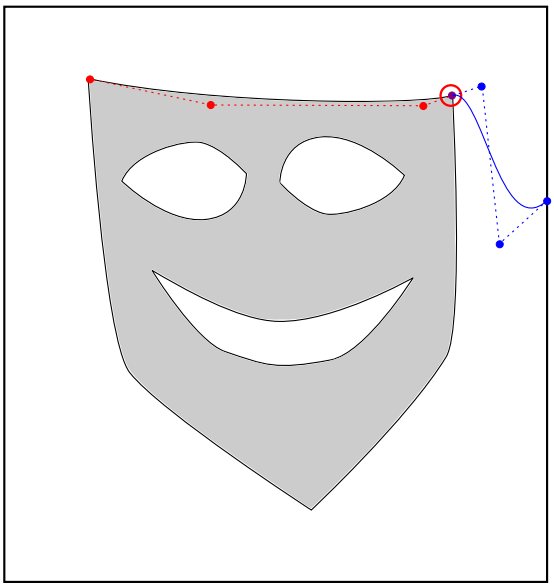


Figure 6: An example of how the puzzle curves are extended to the puzzle enclosure by a C^1-continuously fitting cubic Bézier curve.

curve ends (ECEs). These curve ends are extended to the puzzle enclosure by a single cubic Bézier curve, that fits to the chain C^1-continuously. An example is shown in Figure 6.

The advantage of this constraint is that it limits the complexity of the output to a constant factor times the number of puzzle curves. The choice for extension by a single Bézier curve was motivated by early experimentation revealing that a generated solution would already become quite complex with this number of curves added.

4.3 Parameterisation

The output structure described in the previous section consists of a set of Bézier chains with extendable curve ends (ECEs) that are extended to the border by a single cubic Bézier curve that fits C^1-continuously to the chain. In this section, the corresponding solution space is parameterised, to allow the problem to be solved using parameter optimization techniques.

The parameterisation is on a per-curve-end basis. Take an ECE of a puzzle curve c. This curve end will be extended by a cubic Bézier curve that fits to it with C^1 continuity. Let Q be the Bézier curve that is the last in the chain on this end of c. Q is an n-th order Bézier curve (n can be 1, 2 or 3), described by its control points $q_0, \ldots, q_n$. It will be connected to the puzzle enclosure by cubic Bézier curve B, described by b_0, b_1, b_2 and b_3. We will show that B has three degrees of freedom, which will be expressed by the parameters p_0, p_1, p_2. Firstly, B must start where Q ends. Assuming Q is directed so that q_n is the endpoint of the chain:

$$b_0 = q_n.$$

Secondly, the first derivatives of Q and B must be equal where they join to make them C^1-continuous. Let $Q(t)$ and $B(t)$ be parameterisations of the two curves with $t \in [0, 1)$, for which the curves meet at $Q(1) = q_n = b_0 = B(0)$. This requirement can then be written as

$$Q'(1) = B'(0),$$

which can be written in terms of the control points as

$$b_1 = q_n + (q_n - q_{n-1}).$$

The third control point of B has two degrees of freedom, which we parameterise as the x- and y-value of this point:

$$b_2 = \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}.$$

Finally, the fourth control point has one degree of freedom. This point must lie on the puzzle enclosure, because it is the endpoint of B. Let $R(t)$ be an arc length parameterisation [17] of the rectangular puzzle enclosure curve with $t \in [0, 1)$. We parameterise the fourth control point as follows:

$$b_3 = R(p_2), p_2 \in [0, 1).$$

Puzzle enclosure constraint According to the constraints for a valid Curved Nonogram, we must make sure that B is contained in the puzzle enclosure. An efficient way to do this uses the Convex Hull property [18] of Bézier curves, which states that a Bézier curve is enclosed by its convex hull. We will force the convex hull of the control points of B to be enclosed by the puzzle enclosure. Because the puzzle enclosure is convex (a rectangle), this constraint is satisfied if all of the control points of B lie within the puzzle enclosure.

This constraint is already satisfied for b_0, which lies on an input curve, and b_3 which lies on the puzzle enclosure itself. Point b_1 can lie outside the puzzle enclosure according to the definition given above, depending on the position of q_2 and q_3. In case this happens, we instead use the following formula:

$$b_1 = q_n + a(q_n - q_{n-1}),$$

where a is chosen so that b_1 lies on the puzzle enclosure. In this case, B is no longer a C^1-continuous extension of Q, but is still a G^1-continuous [16] extension, because the directions of the derivatives of the curves in their meeting point b_0 are still the same.

Finally, to ensure that b_2 lies within the puzzle enclosure, we limit p_0 and p_1 in the following ways:

$$p_0 \in [R_{xmin}, R_{xmax}],$$
$$p_1 \in [R_{ymin}, R_{ymax}],$$

where R_{xmin}, denotes the x value of the left side of the puzzle enclosure, etc.

Total number of parameters As shown above, the cubic Bézier curve extension of one curve end is described by three parameters. Since each puzzle curve can have between zero and two ECEs, a solution with n puzzle curves is described by at most $6n$ parameters.

4.4 Implementation details

This section contains implementation details that are not part of the design of the algorithm, but may be important to know for anyone wishing to implement the algorithm for themselves.

Polyline approximation The most important implementation detail is that the arrangements of Bézier curves described in the previous sections are in fact approximated by arrangements of polylines. The reason for this approximation is that the code for the exact representation of Bézier curve arrangements was too unreliable, and also slower than the polyline approximation. Note that the curves read from the input and written to the output are still Bézier curves: they are only approximated when an arrangement is needed to be scored by the evaluation function.

The approximation that we use for a Bézier curve works as follows. First, we make sure that the Bézier curve is parameterised in the standard way. For a Bézier curve B with control points $b_0 \ldots, b_n$, this is defined as:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i b_i, t \in [0, 1].$$

We then sample $B(t)$ at values of t uniformly spread between 0 and 1. These sampled points are the points of the approximating polyline, and we call the number of samples *min_approx_points*. The advantage of the given parameterisation $B(t)$ is that these uniformly sampled points lie closer together where the curvature of B is high, which is where we need the most samples to be accurate.

In the last step of the approximation, we ensure that the distance between two subsequent sampled points is never larger than the pre-defined *max_segment_length*. If two subsequent sampled points $B(s_k)$ and $B(s_{k+1})$ do not meet this criterion, we add the sample point $B(\frac{s_k+s_{k+1}}{2})$ between them. This is done recursively until each pair of subsequent points fulfils the criterion. The reason for this second step is that we want to ensure a minimum density of sample points, even at low-curvature parts of B, because they are used by the dilation penalty calculation in the evaluation function, as detailed in the next section.

4.5 Evaluation function

In this section, we define an evaluation function which takes a Curved Nonogram puzzle as input, and outputs a score that represents how good the puzzle is. This function is based on the observations from Section 3.4, in which we gave informal criteria and three measurements that can be used to judge the non-ambiguity and aesthetic merit of a Curved Nonogram.

For the evaluation function, the focus is more on the non-ambiguity of the puzzle than on the aesthetic merit. The evaluation function is based on three penalties: the vertex penalty, the dilation penalty, and the face penalty. For each of these penalties, a higher value indicates a worse puzzle, and a value of zero is the best result possible.

Vertex penalty The first measurement given in Section 3.4 is the minimum vertex distance of the puzzle's arrangement. This is a simple way to measure certain ambiguities. However, preliminary tests showed that this measurement was too global, which does not fit well with our optimization algorithm that mostly makes local changes. Since the minimum vertex distance is typically determined by one pair of vertices, it does not give any useful information for making changes to a curve on which neither of these two vertices lie.

For this reason we use a different measurement for vertex distance, the *vertex penalty* ρ_{vert}. All pairs of vertices that lie too close together contribute to this measurement. The amount that a vertex pair contributes is based on the distance between them. To be precise:

$$\rho_{vert} = \sum_{v,w \in V | v \neq w \wedge d(v,w) < d_{vert}} d_{vert} - d(v,w),$$

where d_{vert} is a constant that denotes the minimum distance that we wish to have between vertices.

Dilation penalty In Section 3.4 we also introduced the local geometric dilation measurement. This measurement is useful because there are two important types of ambiguous situations that cause the dilation of the arrangement to be high. Firstly, curves that intersect at a shallow angle, and secondly, curves that almost touch without actually intersecting. Because this is also a global measurement, like with the minimum vertex distance, we use a variation on this measurement that we call the *dilation penalty* ρ_{dil}.

The calculation of ρ_{dil} is based on the polyline approximation described in Section 4.4. Let Z be the collection of sampled points, i.e. the vertices of the polylines that approximate the Bézier curves. We define the dilation penalty as:

$$\rho_{dil} = \sum_{p,q \in Z | p \neq q \wedge d(p,q) < d_{dil} \wedge \delta(p,q) > \delta_{max}} \rho_{dil}(p,q),$$

where d_{dil} is the minimum distance that we would like to have between points that have a dilation higher than δ_{max}, and $\rho_{dil}(p,q)$ is the dilation penalty contribution of p and q. This contribution is defined as:

$$\rho_{dil}(p,q) = w(p)w(q)[d_{dil} - d(p,q)].$$

The terms $w(p)$ and $w(q)$ are weights that have to do with the granularity of the polyline approximation around p and q. If p has neighbouring vertices p_{n1} and p_{n2} on the polyline, then its weight is defined as:

$$w(p) = \frac{d(p,p_{n1}) + d(p,p_{n2})}{2}.$$

These weights prevent that the total dilation penalty scales with the granularity of the polyline approximation.

The given measurement ρ_{dil} differs from the local geometric dilation measurement in two ways. Firstly, all problematic point pairs contribute to the measurement, instead of only the pair with the highest dilation. This helps the algorithm make local changes that affect the score. Secondly, the penalty for a pair of points with high dilation that lie close together is based on the (Euclidean) distance between the points, instead of their dilation. For such point pairs, the lower the Euclidean distance, the more it seems like the respective curves are touching - an ambiguous situation, while a higher curve distance (the other term of the dilation) would not increase the ambiguity at all.

Face penalty The third and final measurement that contributes to the score is the face penalty ρ_{face}. This measurement does not relate to any of the specific measurements given in Section 3.4, but it does solve the ambiguity caused by very small faces, which is not always solved by the two measurements given above.

Specifically, during early testing the algorithm produced many very small faces of which the vertices were just far enough apart to not contribute to the close-together vertex penalty. Even though these faces are very narrow in the absolute sense, they do not contribute to the high dilation penalty unless they have a very elongated shape. Thus, the small face penalty is used to measure the presence of these types of faces.

The measurement is calculated similarly to the close-together vertex measurement. All faces that are too small contribute to it according to their size:

$$\rho_{face} = \sum_{f \in F | A(f) < A_{min}} A_{min} - A(f),$$

where $A(f)$ is the area of face f, and A_{min} is a constant that denotes the minimum area that a face should have to be considered unambiguous.

Evaluation function The evaluation function that we use for our algorithm outputs a score that is a weighted sum of the three penalties:

$$score = w_{vert}\rho_{vert} + w_{dil}\rho_{dil} + w_{face}\rho_{face},$$

$$w_{vert}, w_{dil}, w_{face} \geq 0,$$

where w_{vert}, w_{dil}, and w_{face} are the weights for the three corresponding penalties. Note that a higher score indicates a worse solution. The best achievable score is zero.

4.6 Optimization algorithm

Now that we have a parameterisation of the solution space and an evaluation function for solutions, we can introduce our optimization algorithm. The algorithm consists of a pre-processing stage, followed by multiple global and local optimization rounds.

4.6.1 Algorithm structure

The structure of the complete algorithm is shown in Algorithm 1. In the pre-processing step, the input curves are cut and the extension curves E are initialized. The parameters p_0 and p_1 of each extension curve are set to a uniformly randomly chosen point in R and a uniformly randomly chosen point on the boundary of R, respectively.

The extension curves are then optimized in multiple global and local optimization rounds. The algorithm can be said to use a Hill climbing strategy, which means that the evaluation score either stays the same or improves after each global or local round. Finally, the output puzzle is created and returned.

In all the pseudo code in this section, constants like *global_rounds* are shown in cursive with underscore spacing.

4.6.2 Global optimization

The optimization starts with multiple rounds of global Hill climbing, as detailed in Algorithm 2. The optimization happens for one parameter at a time. The parameter is set to several different values spread over its range, while the rest of the parameters stay locked. The value that yields the solution that gets

Algorithm 1 Complete algorithm

```
function ALGORITHM(inputCurves, R)
    puzzleCurves ← CUTCURVES(inputCurves)
    E ← EXTENDRANDOMLY(puzzleCurves)
    for i ← 1 to global_rounds do
        GLOBALOPT(E)
    end for
    for i ← 1 to local_rounds do
        LOCALOPT(E)
    end for
    puzzle ← BUILDPUZZLE(puzzleCurves, E, R)
    return puzzle
end function
```

Algorithm 2 Global optimization round

```
function GLOBALOPT(E)
    score ← EVALUATE(E)
    for all e ∈ E do
        for all p ∈ {0, 1} do
            best = e.getParam(p)
            for all v ∈ GLOBALVALS(p) do
                e.setParam(p, v)
                if EVALUATE(E) > score then
                    score ← EVALUATE(E)
                    best ← v
                end if
            end for
            e.setParam(p, best)
        end for
    end for
end function
```

the best score from the evaluation function is chosen, and the algorithm moves on to the next parameter.

The values that are tested for each parameter are generated by the function shown in Algorithm 3. In this pseudo code, a function *random(a, b)* is used that returns a floating point value from the uniform random distribution over the interval $[a, b)$.

Algorithm 3 Choose global parameter values

```
function globalVals(p)
    vals ← ∅
    if p == 0 then
        steps ← steps_p0
        rangeX ← R_xmax − R_xmin
        rangeY ← R_ymax − R_ymin
        offsetX ← random(0, rangeX / steps)
        offsetY ← random(0, rangeY / steps)
        for i ← 0 to steps − 1 do
            x ← offsetX + i / steps
            for j ← 0 to steps − 1 do
                y ← offsetY + j / steps
                vals.append( [x; y] )
            end for
        end for
    else if p == 1 then
        steps ← steps_p1
        offset ← random(0, 1 / steps)
        for i ← 0 to steps − 1 do
            v ← offset + i / steps
            vals.append(v)
        end for
    end if
    return vals
end function
```

The goal of the global optimization rounds is to quickly move the solution away from the randomized initial state, which is generally very chaotic and ambiguous. Because the tested values are chosen globally, independent of the current values, the influence of the randomness of the initial state is reduced. The random offsets are used to prevent all the third and fourth control points of the extension curves from lying on the same grid. It also increases the usefulness of running multiple global optimization rounds.

4.6.3 Local optimization

The local optimization rounds are similar to the global rounds, in that the parameters are optimized one by one and in that the best scoring value is always chosen. The difference is that the tested values are not chosen globally, but from the neighbourhood of the current value. In case of a p_0 point parameter, the tested values are points that are uniformly spread over a circle centered on the current value of p_0. For a p_1 parameter two values are tested, which are gained by moving a certain distance clockwise or counter-clockwise along the puzzle enclosure R from the current value of p_1. The pseudo code of a local optimization round is shown in Algorithm 4, and the details of how the values are chosen are shown Algorithm 5.

Algorithm 4 Local optimization round

```
function localOpt(E)
    score ← evaluate(E)
    for all e ∈ E do
        for all p ∈ {0, 1} do
            best = e.getParam(p)
            for all v ∈ localVals(p, best) do
                e.setParam(p, v)
                if evaluate(E) > score then
                    score ← evaluate(E)
                    best ← v
                end if
            end for
            e.setParam(p, best)
        end for
    end for
end function
```

The goal of the local optimization rounds is to iteratively make small adjustments to the extension curves that move them away from ambiguous situations. Because only small local changes are made each round, the local optimization round should be repeated a larger number of times than the global optimization.

5 Experiment and results

This section describes an experiment in which the proposed algorithm is used to make Curved Nonograms from a test set of input images. We do this for several different parameter values that determine how the penalties of the evaluation function are calculated. The goal of the experiment is to answer the following two questions:

1. Can simple images be made into unambiguous and visually pleasing Curved Nonograms?

2. Can simple images be made into Curved Nonograms that have a unique solution?

Algorithm 5 Choose local parameter values

```
function localVals(p, v)
    vals ← ∅
    if p == 0 then
        steps ← circle_points
        r ← step_size_p0
        offset ← random(0, 2π / steps)
        for i ← 0 to steps − 1 do
            point ← + i / steps
            for j ← 0 to steps − 1 do
                α ← offset + 2πi / steps
                point ← pointOnCircle(v, r, α)
                vals.append(point)
            end for
        end for
    else if p == 1 then
        size ← step_size_p1
        step ← size / boundaryLength
        vals.append(v − step)
        vals.append(v + step)
    end if
    return vals
end function
```

The set-up of the experiment is described in detail in Section 5.1. Following that, the results of all runs of the algorithm are shown in Section 5.2. The scores of the output are then broken up into their three component parts in Section 5.3. This is followed by Section 5.4 which contains an analysis of the influence of the complexity of the input on the results. We then show a few selected output puzzles in Section 5.5 and discuss the positive and negative aspects of them. Finally, the solvability of the generated Curved Nonograms is discussed in Section 5.6.

5.1 Experiment set-up

Test set To test the performance of the algorithm, we ran a series of experiments on a test set of 16 input images - see Table 1. The images can be found in Appendix A. The number of extendable curve ends for each image is also shown, as this determines the number of parameters that the algorithm has to optimize. The images were created by us, and are meant to be representative for simple black-and-white svg drawings. The drawings are varied in shape and complexity.

Experiment settings For each image from the test set, the algorithm was run with five different settings, which can be found in Table 2. The difference between the settings is the strictness of the evaluation function. Setting (a) is the least strict, which means that only extremely ambiguous situations are taken into account by the evaluation function. The strictness increases from setting (a) through setting (e), which is the most strict.

Image	ECEs
penguin	10
coffee_cup	12
lamb	12
butterfly	14
fish	16
monitor	16
rocket	16
mask	18
airplane	22
flower	22
car	24
hut	24
catface	26
church	32
pumpkin	36
space	38

Table 1: The images that were used as a test set, with the number of ECEs of each image.

Subjectively, setting (b) marks the bottom line for what we find to be visually unambiguous puzzles. Setting (c) corresponds to puzzles that are comfortably unambiguous, they do not really need to be more clear than this. Setting (d) is more strict than necessary for removing ambiguity, but this may improve the aesthetics of the puzzle by forcing a more even distribution of the curves and vertices over the space. Setting (a) and (e) are more extreme variants of (b) and (d) respectively, and serve to test the limits of the algorithm.

	d_{vert}	A_{min}	d_{dil}	δ_{max}	α_{min}
Setting (a)	5	25	3.75	11.5	10°
Setting (b)	8	40	6	7.7	15°
Setting (c)	11	55	8.25	5.8	20°
Setting (d)	14	70	10.5	4.6	25°
Setting (e)	17	85	12.75	3.9	30°

Table 2: The five different settings with which the algorithm was run. α_{min} is derived from δ_{max}: it is the minimum angle at which a curve intersection does not receive a dilation penalty.

Static parameters The parameters of the algorithm that are not included in Table 2 are the same for each of the five settings. Their values are shown in Table 3.

Parameter	Value
min_approx_points	100
max_segment_length	2
w_{vert}	1
w_{dil}	0.04
w_{face}	0.3
global_rounds	3
local_rounds	20
steps_p_0	8
steps_p_1	32
step_size_p_0	10
step_size_p_1	10

Table 3: The parameters of the algorithm that are the same across the five settings.

Performance While performance was not a goal of the algorithm's implementation or design, and it was thus not thoroughly tested, it may still be valuable to give a rough idea of the runtime. Most tests were run on a PC with an Intel®Core™i5-3570 CPU, with four cores @ 3.40 GHz, and 12 GB RAM. With set setup, the algorithm on Setting (c) took 13 hours and 23 minutes to process the *car* test set image, using a single core. The majority of the processing time is spent on the calculation of the dilation penalty. With some optimization of the implementation, we expect it to be possible to cut this time in half, without changing the design of the algorithm.

5.2 Results per experiment setting

Table 4 shows the evaluation function score of the outputs of the test set images for each setting of the experiment. The calculation of the score is explained in Section 4.5. In short: the perfect score is zero, and each ambiguity in the puzzle increases the score. Please note that the scores are not directly comparable between the settings. For example, two vertices that are at 3 distance from one another would incur a close-together vertex penalty of 2 according to setting (a), where $d_{vert} = 5$; the same two vertices would incur a larger close-together vertex penalty of 5 for setting (b), where $d_{vert} = 8$. In general, the same image always receives a higher score according to the evaluation function of a stricter setting, unless the score is zero.

From the score table, we can first of all observe that all images achieved a perfect or very low score (below 10) on setting (a). The same was achieved for all but two images on setting (b), and for 10/16 images on setting (c). For setting (d), there are still 9/16 images that achieved a very low score, but both the number of perfect scores and the average score are much higher than for setting (c). Only 3/16 images have a very low score on setting (e), and many images have a very high score of more than 100 on this setting.

We can conclude that for most images, setting (a) and setting (b) are probably not strict enough, because they achieve a perfect score on these settings, meaning that they could have been optimized further. This is fortunate, because we judge setting (b) to be the bottom line where puzzles become visually unambiguous (if they achieve a perfect score).

A final observation is that there is a large difference between the images in how they score across all settings. This difference seems to be related to the number of extendable curve ends of each image. For example, all images with 18 or less ECEs had a very low score on settings (a) through (d). At the other end of the spectrum, the only images that did not have a very low score on setting (c) were those with 22 or more ECEs. The correlation between the number of ECEs of the input and the score of the output puzzle is further analysed in Section 5.4.

Score progression Table 4 only shows the final scores: the scores of the experiment which correspond to the output puzzles. In order to get a sense of how the score progresses through the rounds of the algorithm, see Figure 7. This figure shows the score progression of four images that are representative for the test set. The graph shows that all scores have more or less converged by the end of the algorithm. In fact, most have converged about halfway through. This lends weight to the scores in Table 4, because it means that they cannot be trivially improved by running the algorithm for a longer period of time.

5.3 Breakdown of the score components

In Section 4.5 we introduced the three components that together make up the evaluation function: the face penalty, the vertex penalty and the dilation penalty. In this section we analyse how much each of these components contributed to the final scores in Table 4. The distribution of the final score over the components is shown in Figure 8.

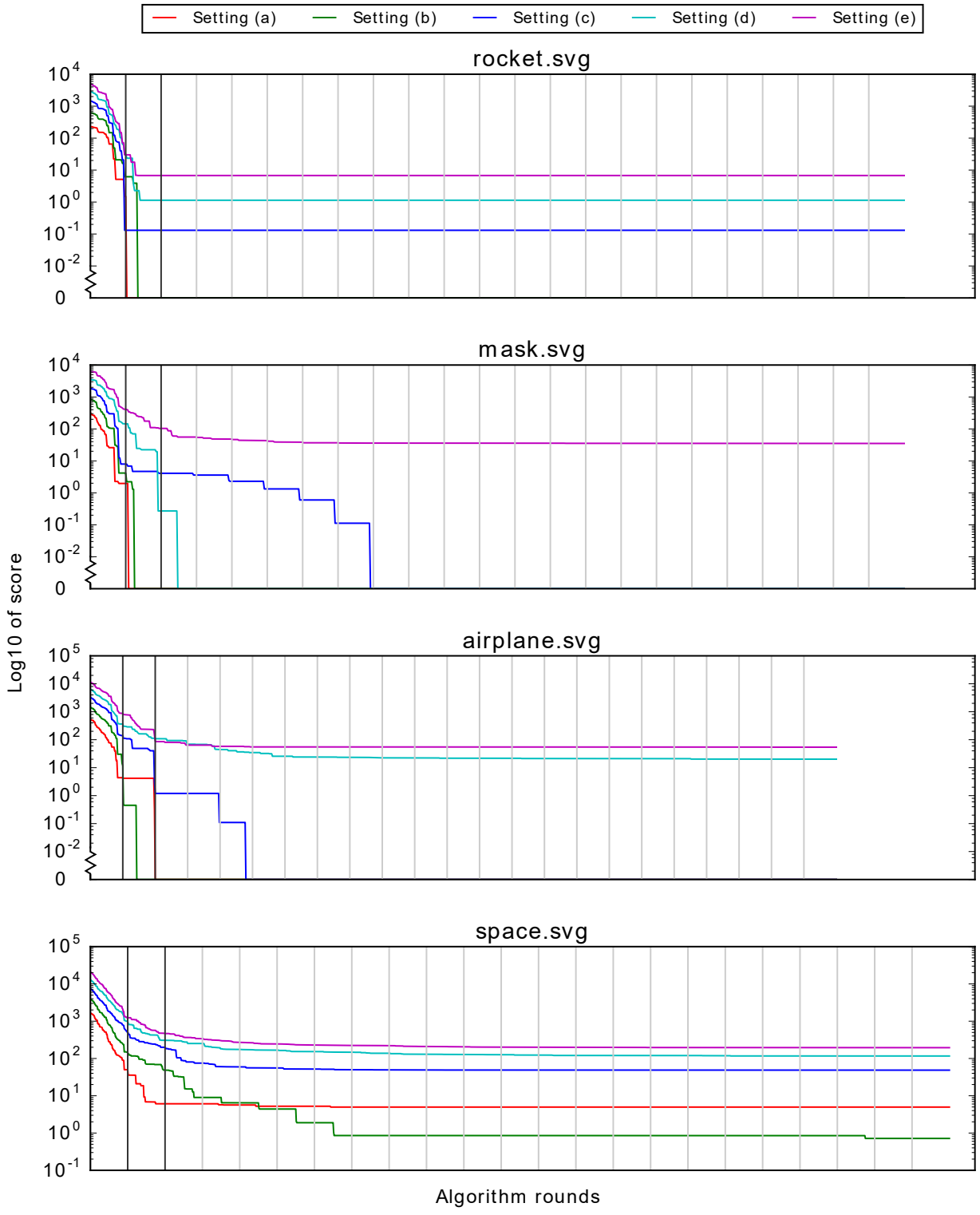


Figure 7: The progression of the score through the rounds of the algorithm of several test set images, for each of the experiment settings. The black vertical lines indicate the start of a global optimization round, and the grey vertical lines indicate the start of a local optimization round.

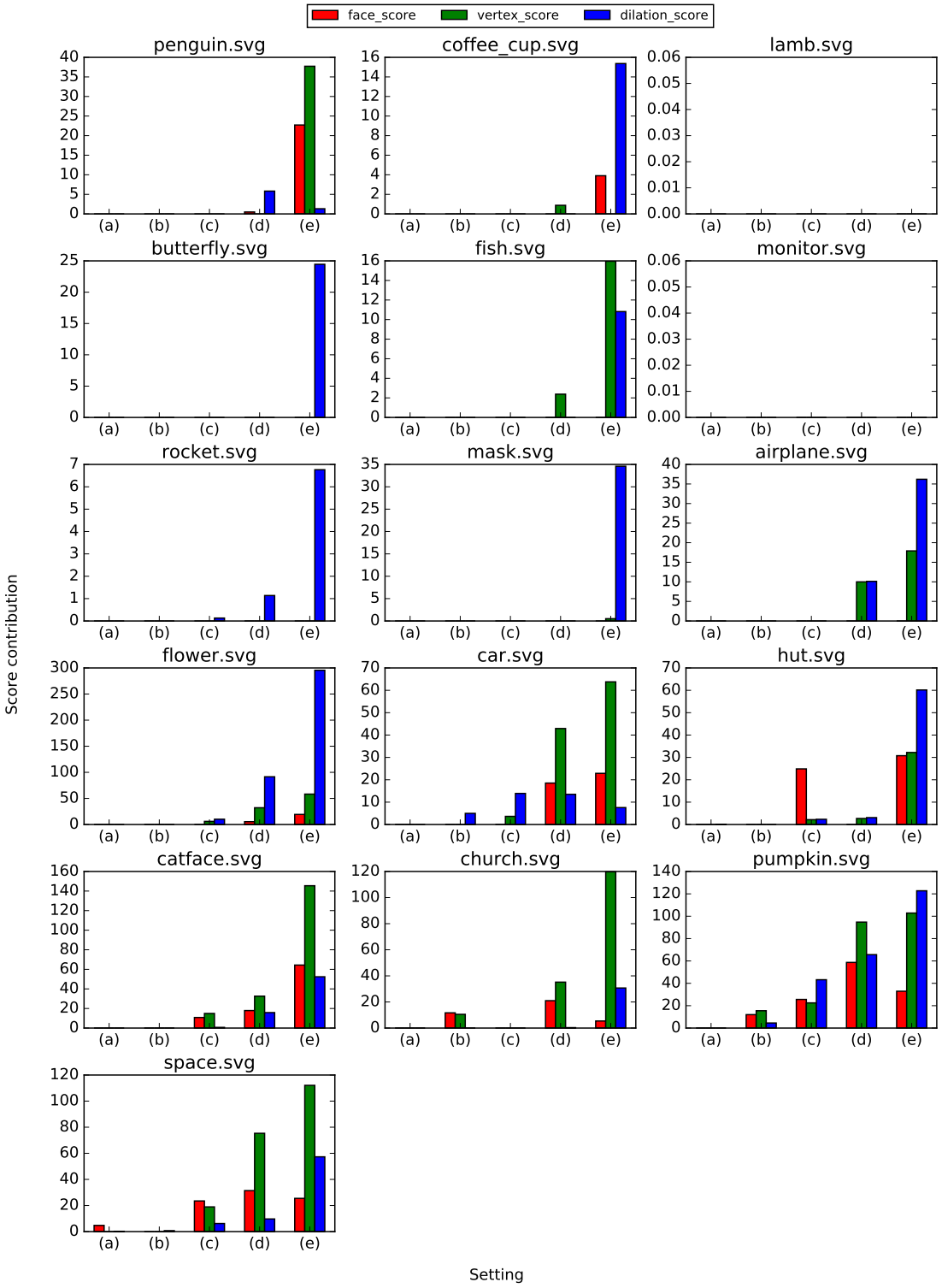


Figure 8: Breakdown of the puzzle's scores into their three components, after they have been weighed. The sum of these components is the total score of the puzzle.

Image	ECEs	Setting (a)	Setting (b)	Setting (c)	Setting (d)	Setting (e)
penguin	10	0.0	0.0	0.0	6.31973	61.7625
coffee_cup	12	0.0	0.0	0.0	0.87834	19.2688
lamb	12	0.0	0.0	0.0	0.0	0.0
butterfly	14	0.0	0.0	0.0	0.0	24.4548
fish	16	0.0	0.0	0.0	2.37818	26.7813
monitor	16	0.0	0.0	0.0	0.0	0.0
rocket	16	0.0	0.0	0.130708	1.1384	6.76382
mask	18	0.0	0.0	0.0	0.0	35.1045
airplane	22	0.0	0.0	0.0	20.1173	54.1086
flower	22	0.0	0.0	16.1588	129.103	373.143
car	24	0.0	4.98756	17.5392	74.9453	94.2376
hut	24	0.100717	0.0	29.3772	5.78301	123.141
catface	26	0.0	0.0	26.4015	66.2723	262.231
church	32	0.0	22.1669	0.0	56.5003	155.865
pumpkin	36	0.0	32.0071	91.3201	219.049	258.504
space	38	4.96683	0.716458	48.6102	116.422	194.859

Table 4: Final score for each test set image for each experiment setting. The number of ECEs is also shown for each image.

The figure shows that the contribution of the score types is relatively well balanced overall. Of the three components, the face penalty is most often absent or relatively low. It can also be observed that most outputs that have a face penalty, also have a vertex penalty that is equal to it or higher. This can be explained by the fact that the vertices incident to a small face typically lie close together, resulting in a vertex penalty. The result is that the function of the face penalty and the vertex penalty partly overlap. The face penalty is mostly useful to prevent the specific case of small, narrow faces that are only incident to two edges and two vertices. These cases are very prevalent if the algorithm is run without a face penalty, which is why we feel that it is not superfluous.

5.4 Relationship between image complexity and score

In this section, we will explore if there is a correlation between the score of the output image and the complexity of the input image. We also give the number of faces in each output puzzle, as a measure of the complexity of the output.

To start off with, Figure 9 shows a point plot for each setting, in which each point represents an output puzzle. The horizontal placement of the point is determined by the number of extendable curve ends of its input image, and the vertical placement shows the score. Note that there are many overlapping points around a score of zero, which unfortunately cannot be seen clearly in the plots.

We expect the number of ECEs of an image to be a reasonable indication of the complexity of an input image, in terms of how difficult it is to handle for the algorithm. It determines the number of extension Béziers that are added to the puzzle, which, assuming that the input image is (almost) unambiguous, are the main cause of ambiguities that the algorithm needs to resolve. Because of this, we expect that there is a positive correlation between the number of ECEs of the input image and the score of the output image.

Correlation Figure 9 also shows the correlation coefficient and regression line for each setting. Because the number of data points in each plot is much too low for statistic significance, we can only pose an hypothesis that may be researched in future work.

Our hypothesis is that there is a positive, but not linear, correlation between the score of the output puzzle and the number of ECEs of the input image. The data seems to support this, as all correlation coefficients are positive, but not very close to 1, and the lines do not fit very well.

The reason that we do not expect a linear correlation is that the score formula is non-linear, and cannot go below zero.

Observations Looking at Figure 9, the test set can be split into three categories. The images with less than 20 ECEs all have very low scores compared to the other images. The only setting for which some

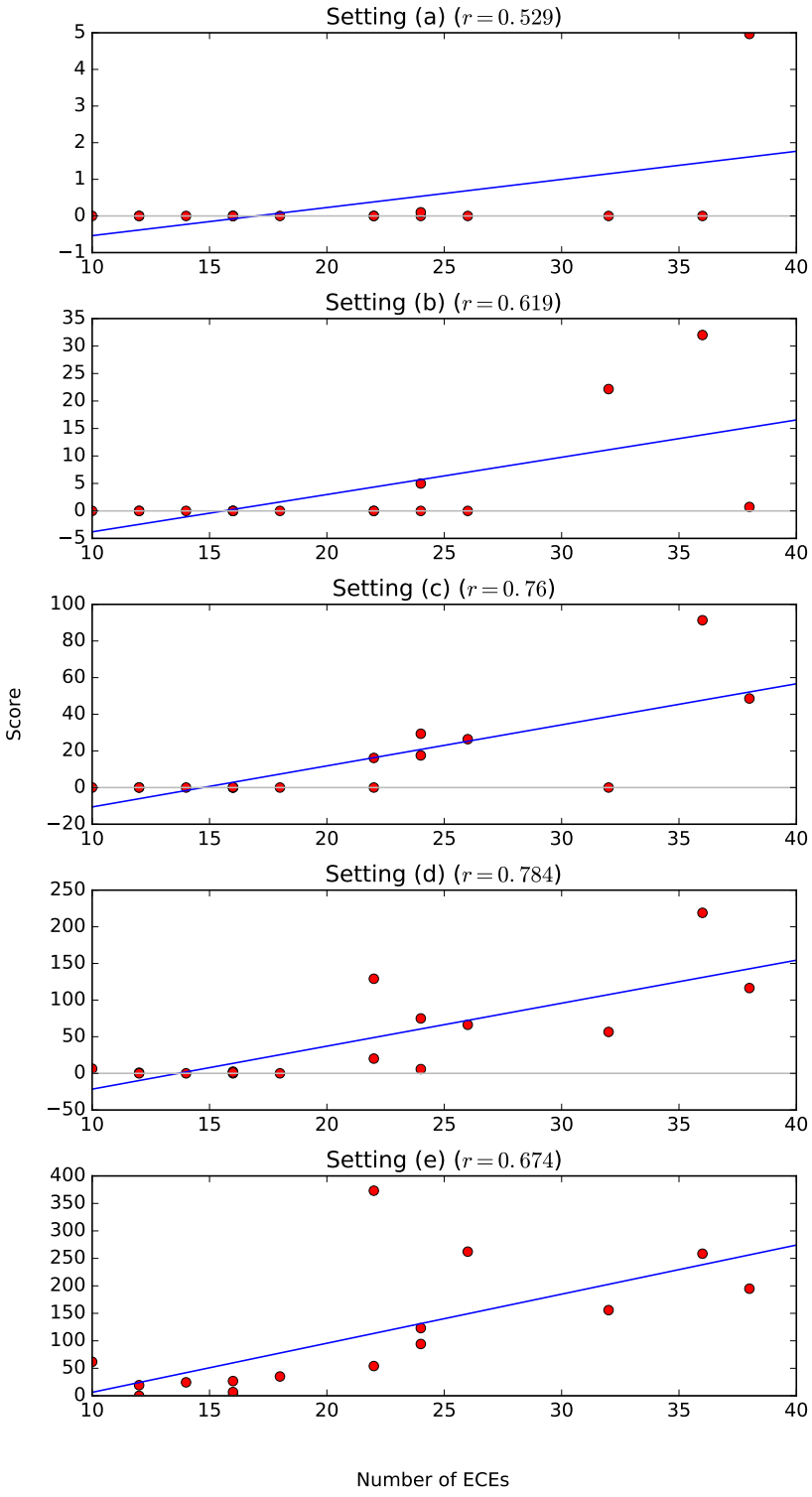


Figure 9: Point plots of the final scores from Table 4 versus the number of ECEs of the input images. The plot of each setting also shows the regression line as well as the correlation coefficient r for the points in that plot.

of them ended with a score significantly above zero is setting (e).

The group of images that have between 20 and 30 ECEs start showing some significant score at setting (c). This group also contains some images that have an exceptionally high score for their number of ECEs, most notably *flower* and *catface*. These images are difficult to handle for the algorithm, for other reasons than their number of ECEs. Both of these images have many ECEs that lie close to one another, e.g., where the petals and leaves of the flower meet in the center, and at the multitude of facial features in *catface*. These cases show a weakness of the method of extending each ECE to the boundary. When ECEs lie close together, and oriented towards each other, it may be a better idea to connect them to each other instead.

Finally, the group of images with more than 30 ECEs has cases where even setting (b) causes a significant score. It also has a lot of images with high scores overall, and a very low percentage of (near) perfect scores.

The differences between these groups suggest that the algorithm is limited in the number of ECEs in an input image that it can successfully handle, depending on the setting. Note that the input images in the third group, although they are the most complex ones of the test set, still look relatively simple. This strongly suggests that a different method is required to handle complex images. Again, a similar method that connects some of the ECEs to each other instead of to the puzzle enclosure is a possible solution to handle images with many ECEs.

Output complexity Table 5 shows the number of faces of each of the generated output puzzles. The numbers range from 40 to 239, with an average of 112 faces (rounded). We can compare these numbers to the number of faces that we typically see in classic Nonograms. In our experience, the classic Nonograms found in puzzle books typically have a 10x10 grid (100 faces), a 15x15 grid (225 faces) or a 20x20 grid (400 faces). We can observe that most of the generated Curved Nonograms have a number of faces that is close to that of a 10x10 or 15x15 classic Nonogram. Of course, these numbers are not directly comparable, but they suggest that the numbers of faces are in the right range for the puzzle to be fun.

Image	ECEs	(a)	(b)	(c)	(d)	(e)
penguin	10	52	43	40	48	48
coffee_cup	12	74	80	62	58	52
lamb	12	59	60	63	54	49
butterfly	14	69	63	59	52	47
fish	16	137	104	88	93	67
monitor	16	114	95	109	78	84
rocket	16	112	108	100	92	80
mask	18	137	103	96	78	80
airplane	22	127	119	107	106	88
flower	22	167	136	157	118	130
car	24	145	124	113	103	110
hut	24	128	108	114	94	95
catface	26	153	131	147	117	130
church	32	184	159	137	127	121
pumpkin	36	237	210	212	209	172
space	38	239	206	190	221	194

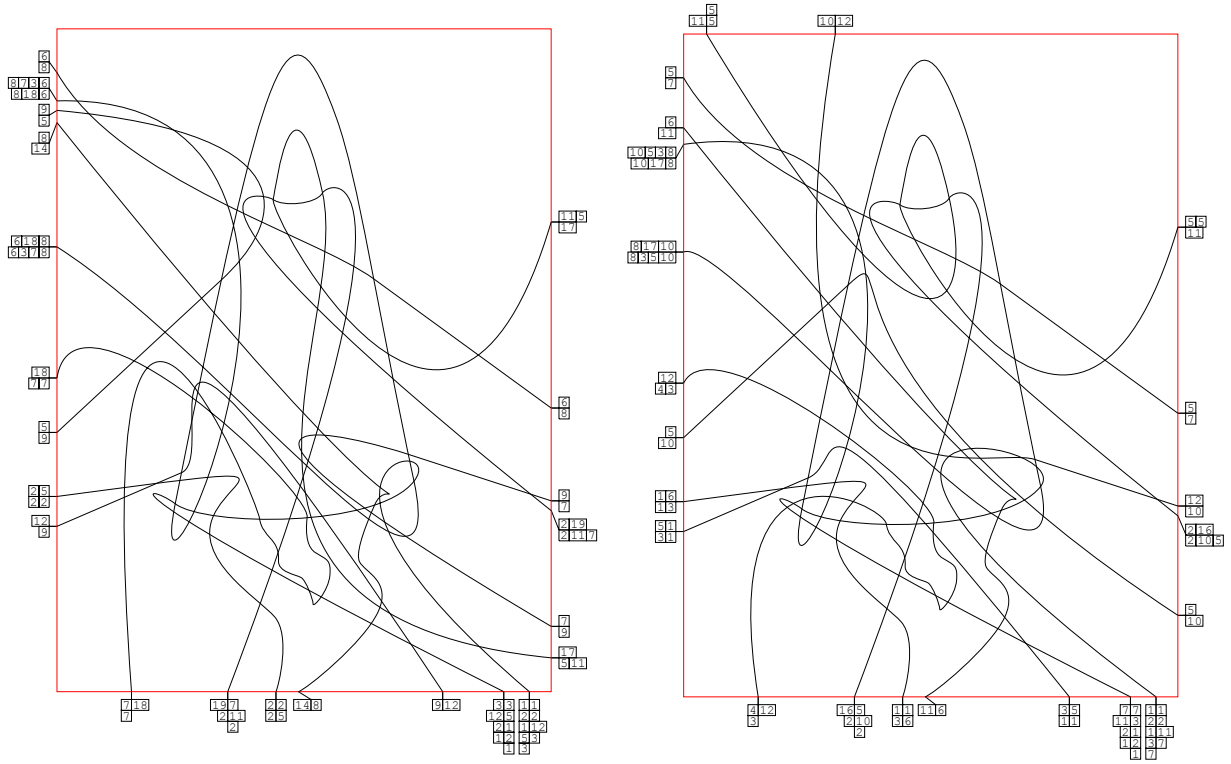
Table 5: The number of faces in each of the generated output puzzles, for settings (a) through (e).

5.5 Qualitative analysis of output puzzles

This section contains several output puzzles from the experiment. Because there is not enough room to show all 16 times 5 output images, we focus on a subset of output images that most clearly show the workings and shortcomings of the algorithm. To get a better impression of the full set of images, please refer to Appendix B.

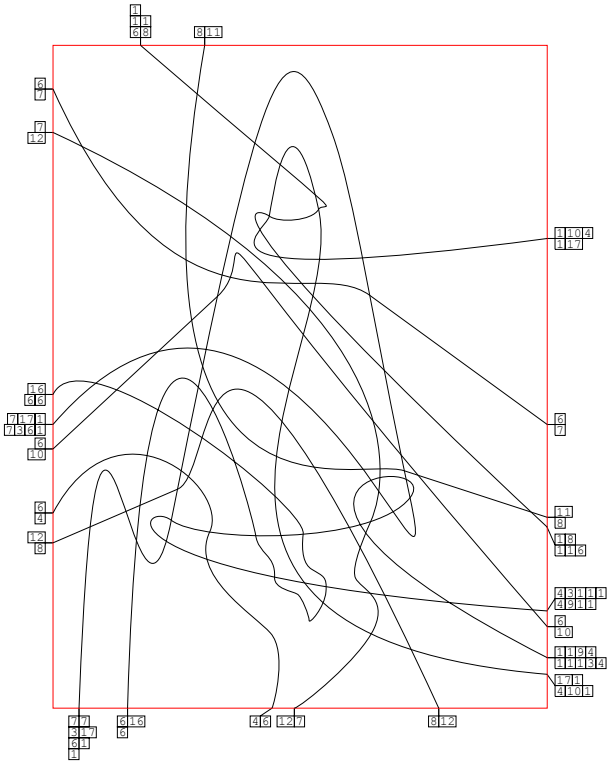
rocket The first image that we will discuss is *rocket* (see Figure 12, of which the output for all five settings is shown in Figure 10 and 11). This image had a very low score for all settings, making it a good example of an image that the algorithm can handle well. In fact, the only penalty that the image receives is a dilation penalty on settings (d) and (e) which cannot be helped by the algorithm. These penalties are applied because an input curve intersects the puzzle boundary at an angle that is too small for settings (d) and (e). This shows a potential drawback of our approach to not modify the input curves: ambiguous situations that only involve input or boundary curves cannot be improved.

Another consequence of this approach is that the algorithm cannot cut G^1-continuous curves, not even at points of high curvature. For an example of this, see the tail of the rocket's middle fire trail, in the bottom-center area of the output images. At such a point, it might be desirable to change the input curve slightly and create a point that is not G^1-continuous,

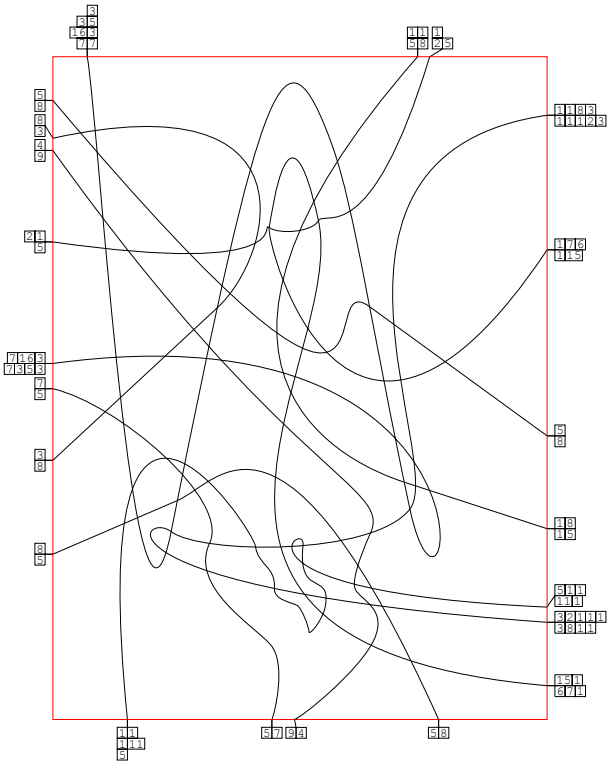


(a) Setting (a)

(b) Setting (b)



(c) Setting (c)



(d) Setting (d)

Figure 10: Output puzzle of test set image *rocket* for settings (a), (b), (c) and (d).

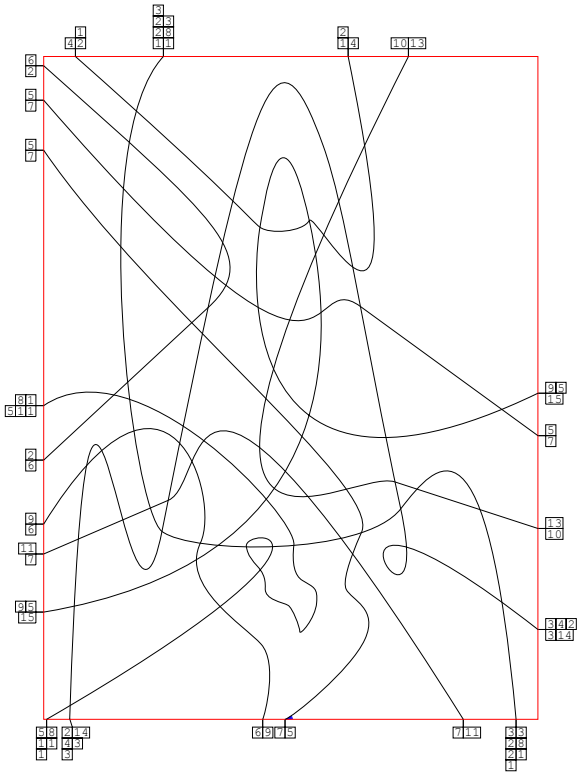


Figure 11: Output puzzle of test set image *rocket* for setting (e).

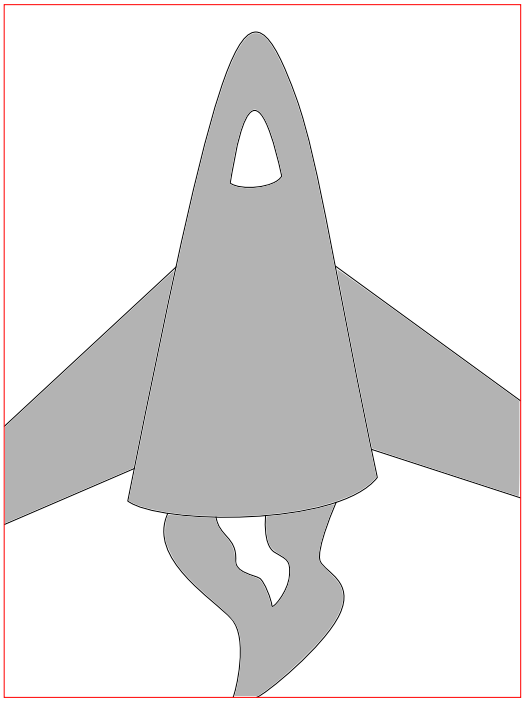
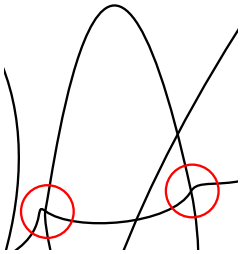
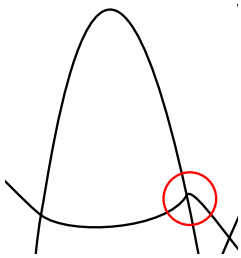


Figure 12: Input image *rocket*.



(a) Setting (d)



(b) Setting(e)

Figure 13: Close-ups of Figure 10d and Figure 11 focused on the window of the rocket. The red circles highlight ambiguous bends in the curve very close to an intersection.

so that it can be cut.

The output puzzles of *rocket* are also a good example of how the outputs are shaped by the different settings. While the initial curve-extension parameters were randomized, we used the same seed for each of the settings. This means that any difference between the output images is purely the result of the strictness of each setting. To start off with, Figure 10a looks quite ugly, and contains some very small faces that are hard to see.

Figure 10b has some curves that are in exactly the same place as the first figure, but those in the most ambiguous regions have been moved by the algorithm. This is most visible at the lower center region of the image, where there are a lot fewer intersections close together.

In Figure 10c a lot of intersections and small faces have disappeared, which helps both the aesthetic quality and the ambiguity. This makes it the first of the images that looks like an enjoyable puzzle. At the upper center region of the image, where the window of the rocket is, are three strange looking faces that consist of one curve pulled slightly over another curve. As explained in Section 5.3, these are a good example of the face penalty at work, which is the reason that they have not been made too small by the algorithm. However, from an aesthetic perspective, the puzzle would look better if it had fewer of these types of faces. These kinds of faces are quite prevalent in the output, because it is hard for the Hill climbing algorithm in the local phase to pull the two curves across one another and make the face disappear. A small step in this direction will at some point incur a face penalty, and possibly a vertex or dilation penalty as well, causing it to not be chosen by the algorithm.

Figure 10d seems to be a relatively smaller step up. The last faces that might be considered too small are gone, and the curves are spread more equally over the image. Finally, Figure 11 is the output of the strictest setting. Compared to the other outputs, it looks very clean and aesthetically pleasing.

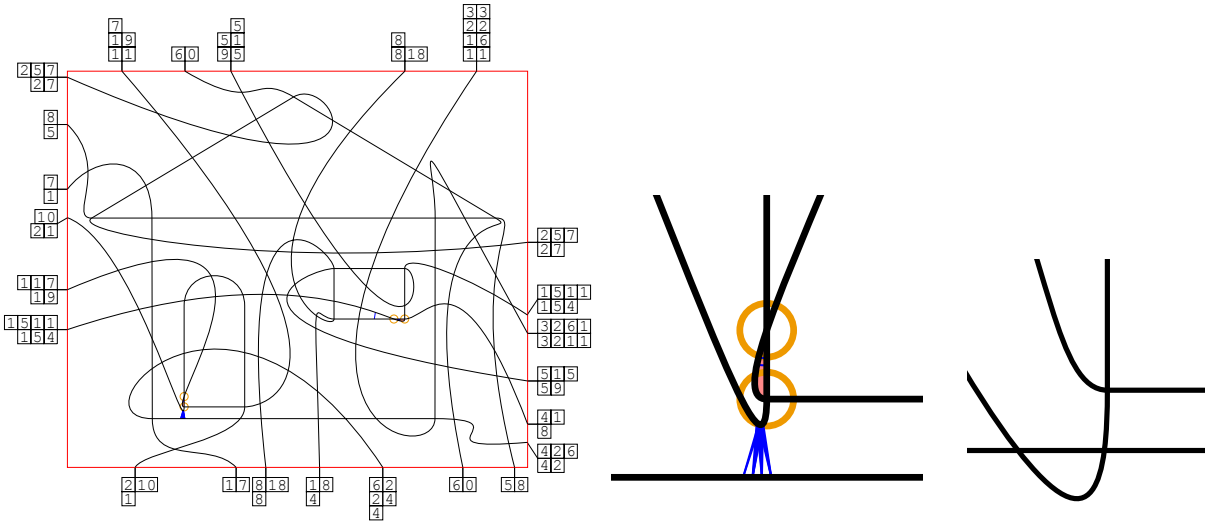
On the subject of aesthetics, one of the weak points of these five output puzzles is that they all contain a number of large faces that are not very interesting. This is most visible in the top right corner of each of the output puzzles. The measurement m_b that we introduced in Section 3.4 - a measurement for the largest inscribed circle of any face - can be used to this aesthetic weak point. This measurement is not included in the algorithm because of the focus on removing ambiguity over improving aesthetics. However, it could be included in future work that focuses more on the aesthetic aspects, to penalise large, empty faces such as these.

Curve bends near intersections The final observation that we wish to make about the *rocket* puzzles is shown in Figure 13. This figure shows a close-up of the area around the rocket's window for the output puzzles of setting (d) and setting (e). In these close-ups, a total of three more or less ambiguous areas are shown. The problem with each of these is that there is high curvature (a bend in the curve) very close to an intersection. This type of ambiguity appears in multiple output puzzles, and is important to highlight because it is the only type of ambiguity that did not receive a penalty from the evaluation function.

First of all, this shows a weak point in the evaluation function, since it should penalize all ambiguities that can appear. Secondly, there are two reasons why this type of ambiguity is created relatively often by our algorithm. The first of these reasons is that it is encouraged by the dilation penalty. Especially on the stricter settings, curves need to intersect at large angle, which can be achieved by bending the curve just before the intersection. The second reason is that our algorithm has no control over the placement of the second control point of an extension Bézier curve, because we enforce C^1-continuity. This leads to many cases where the second control point lies very close to the intersection point (which is the first control point), which can result in very high curvature around these points. A solution would be to only enforce G^1-continuity, which gives the second control point one degree of freedom. This degree of freedom can either be used to simply place the second control point a set distance away from the first control point, or it could be given as an extra degree of freedom for the algorithm to optimize.

hut Figure 14a shows the output puzzle of the *hut* image on setting (c). This is a good example of a puzzle with a non-zero score that is easy to correct by hand. The puzzle contains two ambiguous areas, one on the lower left and one just right of the center, both of which are the result of the algorithm getting stuck in a local minimum.

We focus on the lower left area, shown enlarged in Figure 14b. This area contains one face that is too small, coloured pink; two vertices that lie too close to each other, marked with yellow circles; and a number of sampled point pairs that lie too close while having dilation that is too high, marked by blue lines. This situation is a local minimum in the evaluation function. First of all, some of these curves are part of the



(a) whole puzzle (b) zoomed in (c) possible fix

Figure 14: Output puzzle of test set image *hut* on setting (c). The image on the right shows the ambiguous area around the bottom left at a larger scale. Ambiguities that contribute to the score have been marked.

input drawing (see Appendix A), which means that they cannot be manipulated. The curve that goes towards the top left cannot easily be moved out of the way, because this would increase its dilation with the bottom curve. It would be ideal to make it go straight down and cross the bottom curve, but any incremental changes in that direction have the same problem of increasing the dilation. Making the small face more elongated to separate the vertices increases the dilation within the face. Enlarging the face towards the left would introduce a dilation penalty with the curve left of it.

This situation is easy to fix by hand, for example see Figure 14c. This fix was created simply by changing the position of the third control point of two extending Bézier curves, which solved all of the ambiguities highlighted in Figure 14b without introducing them anywhere else in the puzzle. The conclusion that can be drawn from this is that these situations can be fixed by our method of extending curves and manipulating the control points of those extensions, but not by the used search algorithm.

Finally, an observation that can be made when looking at Figure 14a as a whole is that the input image is relatively easy to spot. The *hut* image consists mostly of straight lines, which makes the high-variation curves added by the algorithm stand out. This could be prevented by basing the extension curves on the input curves of the image. For example, the extension curves could be built by copying and scaling (parts of) the input curves.

5.6 Solvability results

In Section 3.3 we defined the four classes of solvability for (Curved) Nonograms. Of the 80 output puzzles that were produced by the experiment, 75 fall in the simple class of solvability. The other 5 puzzles (*fish* on settings (c), (d) and (e) and *car* on settings (b) and (d)) had multiple solutions. These five puzzles all have multiple solutions because they have a closed background curve that is not intersected by any puzzle curves. In this situation, the face inside the background curve is not associated with any descriptions, so in any solution of the puzzle it can be either black or white. Fortunately, this special case can be fixed by adding an additional rule to the Curved Nonogram, that a face should be left white if there is no information available for it. In other words, the user is asked to find a minimal solution, where as few faces as possible are coloured black. Re-running the experiment with this added rule resulted in 80/80 puzzles that were simple.

The fact that all output puzzles are simple, while the algorithm does not take the solvability into account, is a good sign. It suggests that the solvability does not need to be a primary concern when generating these kinds of puzzles, which makes it easier to focus on the aesthetic aspects.

A possible concern is that the produced puzzles may be too easy. Some of the harder puzzle book Nonograms require the puzzler to think a few steps ahead, meaning that they are not simple solvable,

but close to it. Fortunately, making a puzzle harder is easy to achieve. One way to do this is by removing descriptions from the puzzle until it has the desired difficulty. Of course, this must be done in such a way that the puzzle keeps a unique solution, instead of multiple solutions. Combining a pure-aesthetics algorithm like the one we proposed with such a post-processing step may be a good way to generate good puzzles in the future.

6 Conclusion

In this paper we have introduced a new type of pencil-and-paper puzzle called the Curved Nonogram, which is a variation on the existing Nonogram puzzle. Where a classical Nonogram consists of a square grid of rows and columns, a Curved Nonogram is built from arbitrary curves which make it free to take any shape. This property gives the puzzle designer more artistic freedom in shaping the solution image as they want. In addition, much more variety is possible in the shape of the unsolved puzzle, while classic Nonograms only differ in the number of rows and columns. This gives a unique feel to each Curved Nonogram even before it is solved. Both of these differences make Curved Nonograms more visually appealing and interesting. New puzzle mechanics are also introduced. For example, a new situation arises when one side of a curve comes across the same face more than once. This gives a new form of information about the face, which the puzzler will sometimes have to use in order to solve the puzzle.

After introducing the Curved Nonogram itself, we also introduced ways to classify and judge these puzzles. First, we defined the four classes of solvability to which a Curved Nonogram can belong: no solution, multiple solutions, unique solution and simple. Secondly, we gave three measurements to judge its aesthetics and level of ambiguity: the minimum vertex distance, the minimum edge length, the largest inscribed circle of any face, and the local geometric dilation.

Next, we proposed an algorithm that takes a coloured line drawing as input, and makes a Curved Nonogram whose solution image matches the drawing. The algorithm cuts the input curves where they are not smooth and extends the resulting curve pieces to the puzzle enclosure, which results in a valid Curved Nonogram. The goal of the algorithm is to make these extensions in a way that minimizes the ambiguity in the puzzle. This is achieved by introducing an evaluation function that measures ambiguity in the puzzle. The evaluation function has three terms, based on some of the three measurements defined earlier: the vertex penalty, the dilation penalty and the face penalty. A simple optimization algorithm is used to find a puzzle with a low evaluation score (lower is better), ideally zero.

Finally, we conducted an experiment in which we ran the algorithm on a test set of 16 simple images, for five different settings of the algorithm. The settings impacted the level of ambiguity that the algorithm would tolerate, ranging from very forgiving (setting (a)) to very strict (setting (e)). We found that a very low score (under 10) was achieved for 14/16 of the images on setting (b), for 10/16 images on setting (c) and for 9/16 images on setting (d). This means that a nice puzzle was created for at more than half of the images, and that only two images did not achieve a good result on setting (b), which results in only barely unambiguous outputs. We also found that the scores of the output puzzles seemed to have a positive correlation with the number of extendable curve ends of the input images, although our data sample was really too small for proper analysis.

In the last part of the experiment we classified the output puzzles by solvability, and found that 75/80 puzzles were simple. The other five are a special case of multiple solutions: they contain faces that are only adjacent to background curves, which means that they have no descriptions associated with them, so they could be either colour. While solvability was not a concern of the algorithm, it is good to know that the puzzles produced this way are likely to be simple. This is the desired solvability class for all but the hardest puzzle book puzzles, and making them harder to solve can be as easy as removing some of the descriptions.

In conclusion, by introducing Curved Nonograms together with a generation algorithm for these puzzles, we were able to produce a set of puzzles that may serve as an example of what this kind of puzzle looks like. While the generated puzzles are not perfect, they are at least mostly unambiguous and simply solvable. In our opinion, the produced puzzles look interesting enough that we hope that the Curved Nonogram concept is expanded upon in the future. With this goal in mind, we have presented many ways in which our algorithm can be adjusted and improved, which are summarized and expanded upon in Section 7.

7 Future work

Since the Curved Nonogram is a new type of puzzle, and the proposed algorithm is the first algorithm that

generates this type of puzzle, there are many improvements and alternatives that could be part of future work. In this section we will repeat the suggestions that are already mentioned in the thesis, and add new ideas as well.

Pre-processing the input First of all, there are many other ways to do the pre-processing of the input image. If the algorithm is allowed to change the input curves, it would be possible to cut the curves at G^1 continuous points that have high curvature. This may look more natural, because those points often do not look smooth at all. In this case, the curve should be changed slightly to create a cutting point that is not G^1-continuous, so that the separated curves intersect at an angle greater than zero.

Another possibility would be to give the algorithm the power to change all the input curves, using force directed placement like [13] uses for Graph drawing to encourage but not enforce that the curves stay similar to the input.

An option that is closer to our algorithm is to let the optimization algorithm change the input curves until they have a score of zero, before extending them. This would solve the problem that we found where ambiguous input curves made it impossible to achieve a perfect score.

The initial extension curves An area in which big improvements can be made is how the curves are initially extended, i.e. the start state of the optimization algorithm. Our method of spreading the control points of the extension Béziers uniformly random creates extremely chaotic start states, which makes it harder to optimize. The first step to improving the extension curves would be to enforce a minimum distance between the first and second control point. The experiment outputs show many cases where ambiguity is caused by high curvature at the start of the extension curve. A bigger change to avoid the chaos is to extend each curve to the puzzle enclosure with a straight line. A variation on this is to use a randomized Bézier curve that is allowed to deviate a set amount from that line.

Another option that we discussed is to copy pieces of the input curves, and use those to build the extending curves instead of always using a cubic Bézier curve. Although this is a more complicated process, the advantage is that the solution image should be better hidden in the puzzle, because the properties of the generated curves will be similar to those of the input curves (e.g., many straight lines and 90° angles).

The algorithm can also be changed to handle inputs with many extendable curve ends better, by connecting some of those curve ends to each other instead of to the boundary. In this variation, an extra step could be added to the algorithm in which it determines the best pairs of ECEs to connect, based on smoothness, distance or the resulting ambiguities.

Evaluation and optimization If the previously given suggestion of cutting high-curvature points is implemented, it would also be interesting to add a penalty term for curve pieces with high curvature to the evaluation function. Many of the problems that remained in the output puzzles of the experiment involved such high-curvature pieces.

Another measurement that can be added is the *Largest inscribed circle of any face*, introduced in Section 3.4. We expect that adding (a variant of) this measurement to the evaluation function would greatly improve the aesthetics of the puzzle, by forcing a more equal distribution of the curves over the puzzle. This should especially help for the faces adjacent to the puzzle enclosure, which were somewhat large, empty and uninteresting for many of the generated output puzzles.

For the optimization of the evaluation function, we used a simple Hill climbing algorithm. This often resulted in the algorithm getting stuck in a local optimum. It would be interesting to see how well the algorithm performs with a more sophisticated search heuristic, such as Simulated Annealing. What would be even more interesting to research is if (parts of) the evaluation function could be formulated as a mathematical function and solved analytically. Although this might require a different evaluation function, it would certainly give us more insight into the optimization problem than any search heuristic.

Solvability and difficulty In Section 5.6 we found that the only output puzzles that were not in the simple class of solvability, were those with one or more faces that were not associated with any description. The result is that it can have either colour, which gives the puzzle multiple solutions if it has any. For cases like these, a rule can be added to the Curved Nonogram definition that only a *minimal solution* is considered correct. That is to say, a solution should have the minimum amount of black-coloured faces, and a maximum amount of white-coloured faces. With this rule added, such problematic disconnected faces must be coloured white and the problem is solved.

An interesting study into the solvability of Curved Nonograms would be to see how many descriptions

can be removed from the puzzle curves before the puzzle is no longer simple. This could be used as a tool to bring the puzzle to a desired level of difficulty. This process could even continue if the puzzle has become uniquely solvable but no longer simple, in which case the difficulty could be defined as the maximum search depth (i.e. the maximum number of concurrent assumptions that must be made on the face colours) that is required to find the solution. For such a study, it would be interesting to see how many descriptions can be removed before the puzzle starts to have multiple solutions.

User studies Finally, while we have not touched on the subject of user studies in this paper, there are multiple interesting user studies that can be done on Curved Nonograms. Before Curved Nonograms can be included in actual puzzle books, user studies need to be done on whether the rules and puzzle concept are clear enough for the general public. Similarly, a study could be done to determine the desired level of complexity of a Curved Nonogram in a puzzle book, e.g. in terms of the number of curves, or the maximum number of faces adjacent to one side of a puzzle curve.

A different topic that is also interesting for a user study is how well the solution images of (generated) Curved Nonograms are hidden in the puzzle. Some of the test set images surely seem to be too clearly visible in the output puzzle. It would be especially interesting to find a way to measure how well-hidden the solution image is for a given puzzle. A user study would be needed to find such a measurement and prove that it works.

References

[1] N. Ueda and T. Nagao, "NP-completeness results for NONOGRAM via parsimonious reductions," *preprint*, 1996.

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

[3] C.-H. Yu, H.-L. Lee, and L.-H. Chen, "An efficient algorithm for solving nonograms," *Applied Intelligence*, vol. 35, no. 1, pp. 18–31, 2011.

[4] K. J. Batenburg and W. A. Kosters, "Solving Nonograms by combining relaxations," *Pattern Recognition*, vol. 42, no. 8, pp. 1672–1683, 2009.

[5] S. Salcedo-Sanz, E. G. Ortíz-García, A. M. Pérez-Bellido, A. Portilla-Figueras, and X. Yao, "Solving Japanese puzzles with heuristics," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pp. 224–231, IEEE, 2007.

[6] J. Wolter, "The 'pbnsolve' paint-by-number puzzle solver." Available at http://webpbn.com/pbnsolve.html.

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

[8] E. G. Ortiz-García, S. Salcedo-Sanz, J. M. Leiva-Murillo, A. M. Pérez-Bellido, and J. A. Portilla-Figueras, "Automated generation and visualization of picture-logic puzzles," *Computers & Graphics*, vol. 31, no. 5, pp. 750–760, 2007.

[9] M. Hunt, C. Pong, and G. Tucker, "Difficulty-driven sudoku puzzle generation," *Journal of Undergraduate Mathematics and Its Applications (UMAPJournal)*, p. 343, 2007.

[10] A. M. Smith, E. Butler, and Z. Popovic, "Quantifying over play: Constraining undesirable solutions in puzzle design.," in *Foundations of Digital Games conference, 2013*, pp. 221–228, 2013.

[11] M. Löffler, M. Kaiser, T. van Kapel, G. Klappe, M. van Kreveld, and F. Staals, "The connect-the-dots family of puzzles: design and automatic generation," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 72, 2014.

[12] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[13] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[14] H. C. Purchase, R. F. Cohen, and M. James, "Validating graph drawing aesthetics," in *Graph Drawing*, pp. 435–446, Springer, 1996.

[15] H. C. Purchase, "Metrics for graph drawing aesthetics," *Journal of Visual Languages & Computing*, vol. 13, no. 5, pp. 501–516, 2002.

[16] B. A. Barsky and T. D. DeRose, *Geometric continuity of parametric curves*. Computer Science Division, University of California, 1984.

[17] R. T. Farouki, "Arclength parameterization," *Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable*, pp. 369–380, 2008.

[18] G. Farin, *Curves and surfaces for computer-aided geometric design: a practical guide*, p. 35. Elsevier, 2014.

A Test set

This appendix contains the 16 test set images that were used in the experiment, shown in Figure 15 through Figure 30.

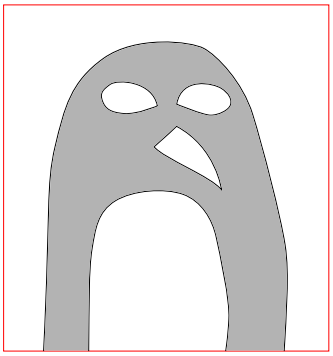


Figure 15: Test set image *penguin*.

Figure 16: Test set image *coffe_cup*.

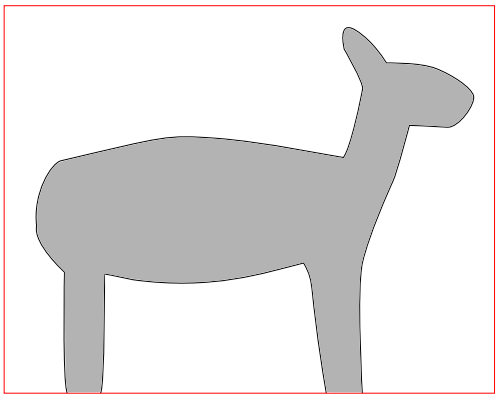


Figure 17: Test set image *lamb*.

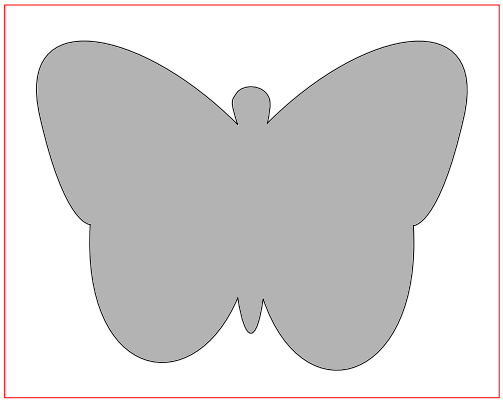


Figure 18: Test set image *butterfly*.

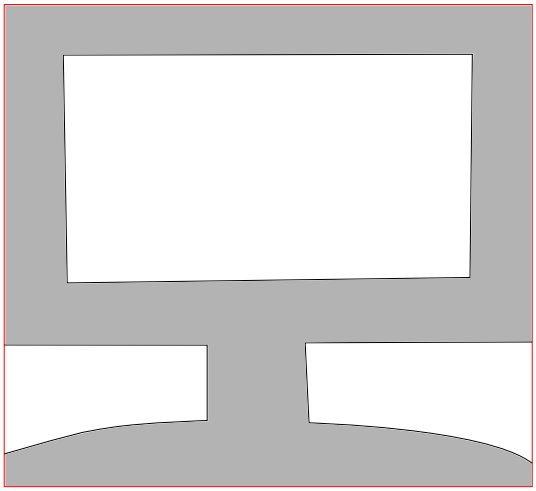


Figure 20: Test set image *monitor*.

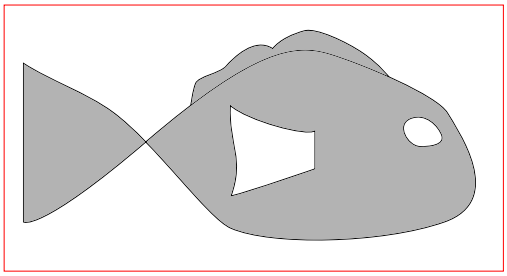


Figure 19: Test set image *fish*.

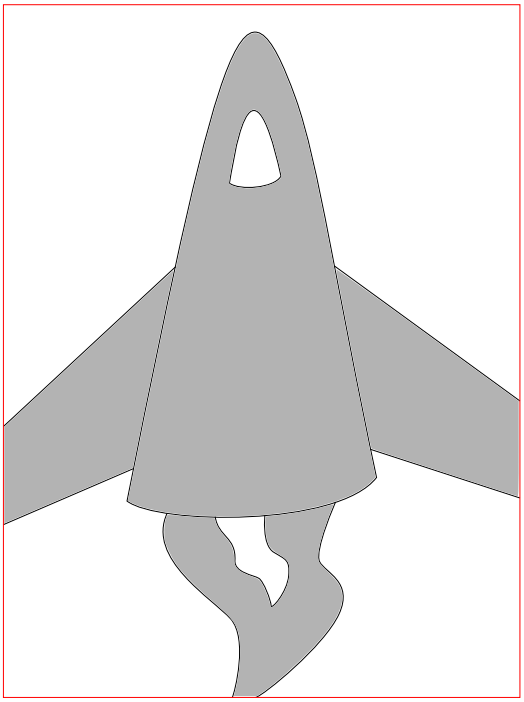


Figure 21: Test set image *rocket*.

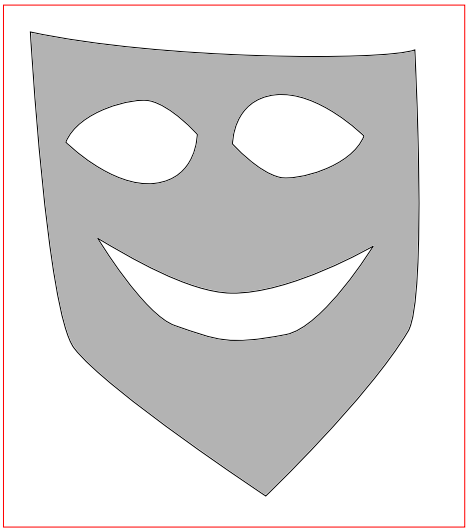


Figure 22: Test set image *mask*.

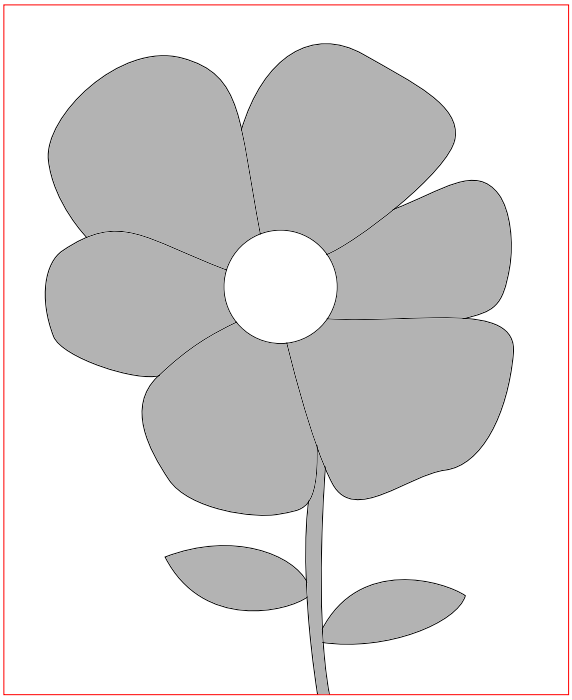


Figure 24: Test set image *flower*.

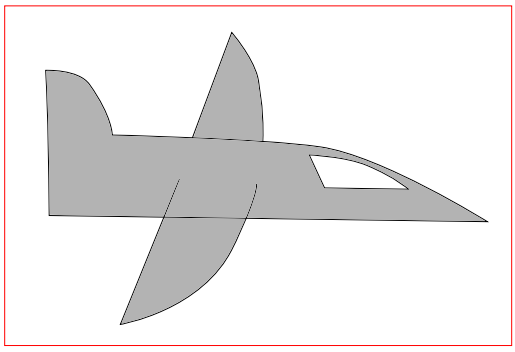


Figure 23: Test set image *airplane*.

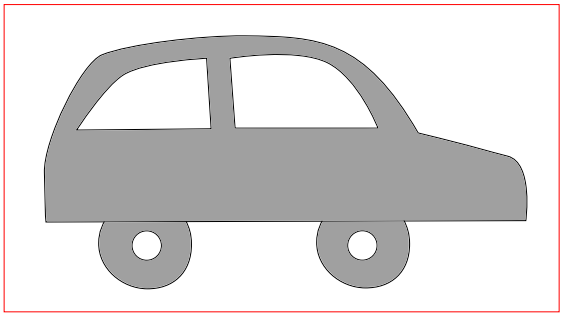


Figure 25: Test set image *car*.

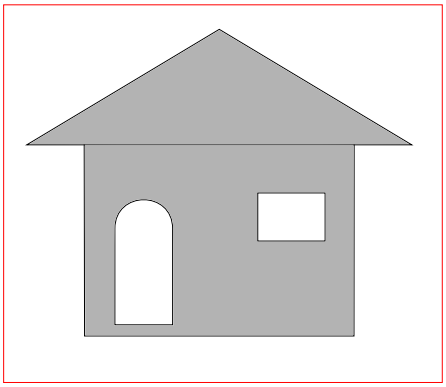


Figure 26: Test set image *hut*.

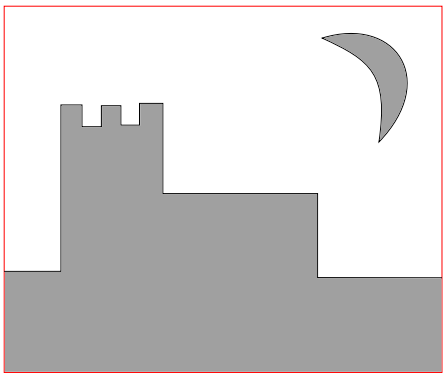


Figure 28: Test set image *church*.

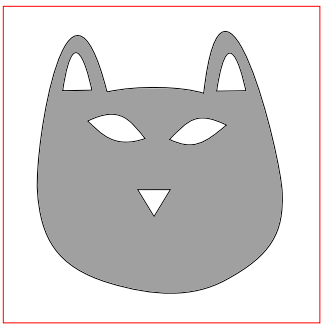


Figure 27: Test set image *catface*.

Figure 29: Test set image *pumpkin*.

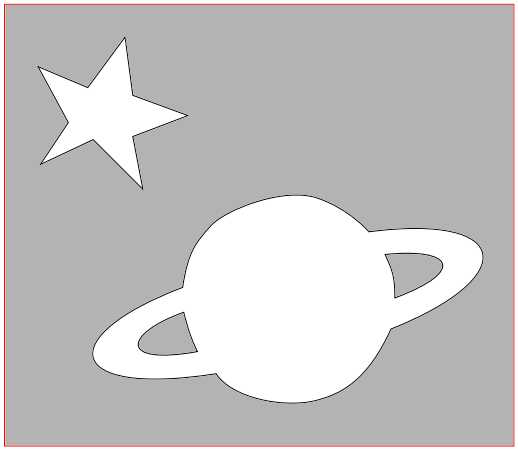


Figure 30: Test set image *space*.

B Experiment output on setting (c)

This appendix contains the output puzzles that were produced in the experiment by running the algorithm on the test set on setting (c). Of each output puzzle, the empty puzzle is shown next to the solution. These outputs are shown in Figure 37 through Figure 46.

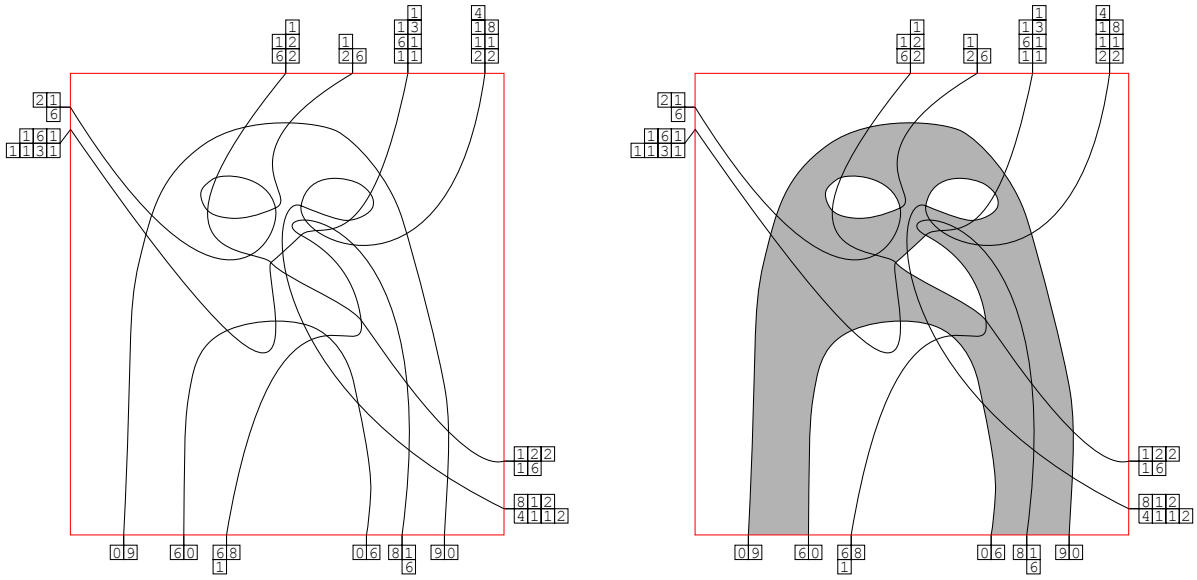


Figure 31: Output puzzle of *penguin* on setting (c).

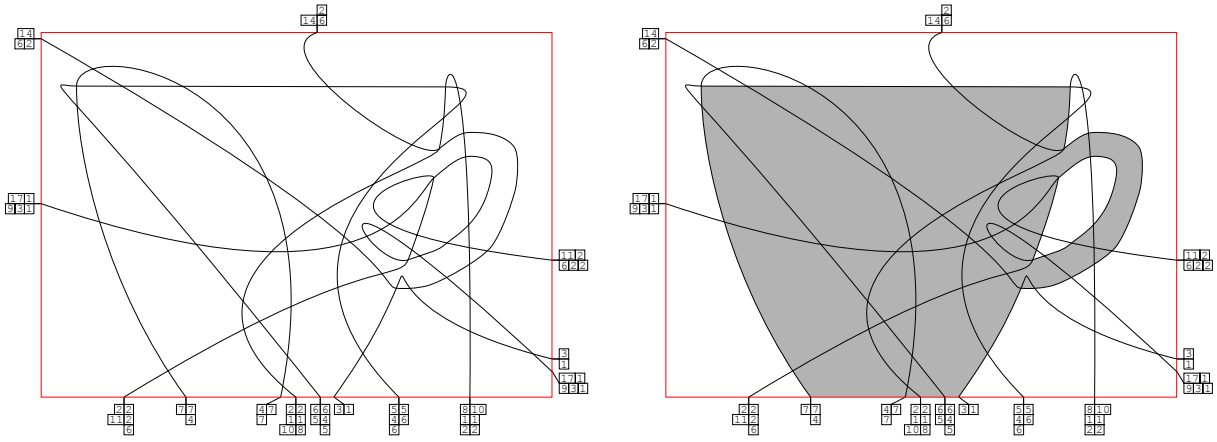


Figure 32: Output puzzle of *coffee_cup* on setting (c).

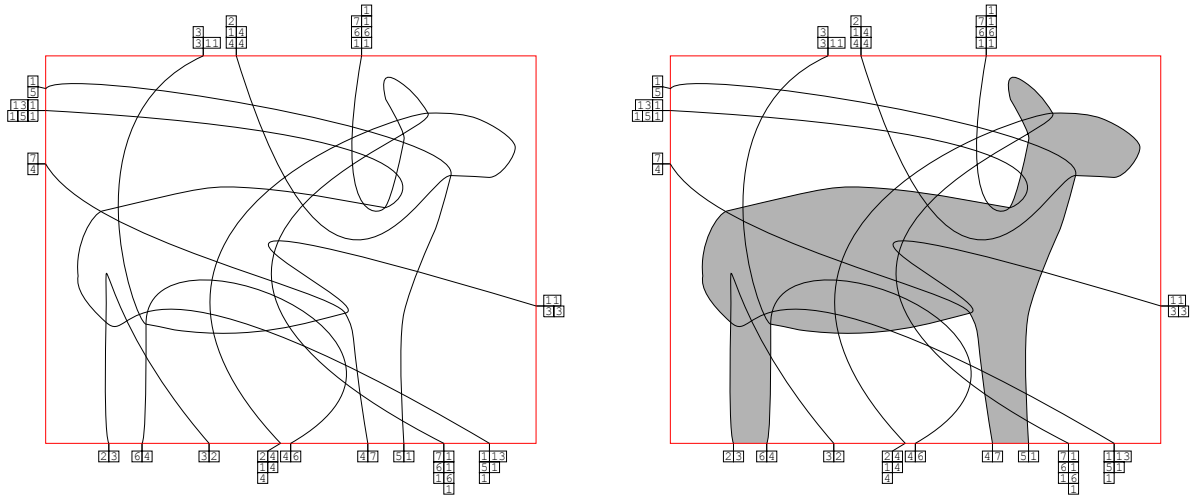


Figure 33: Output puzzle of *lamb* on setting (c).

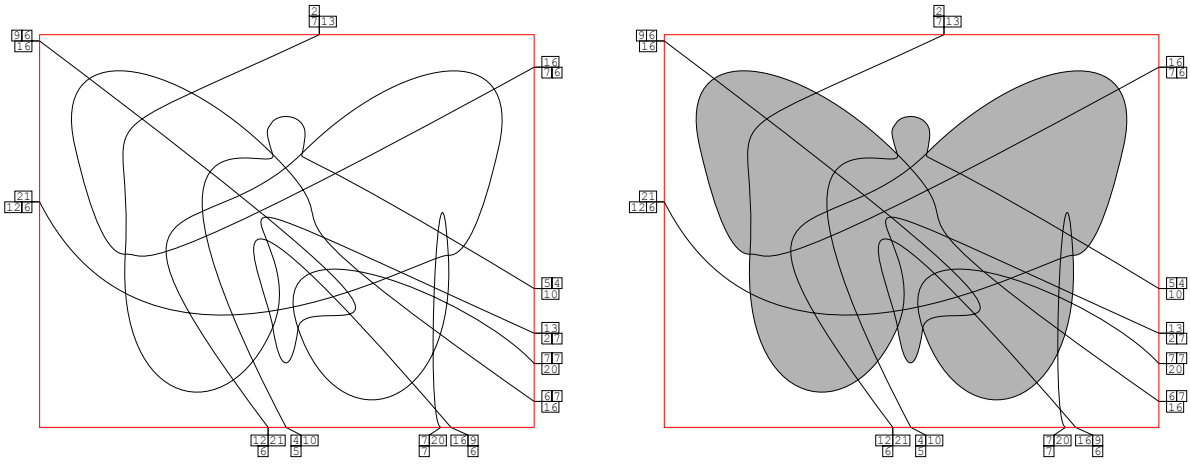


Figure 34: Output puzzle of *butterfly* on setting (c).

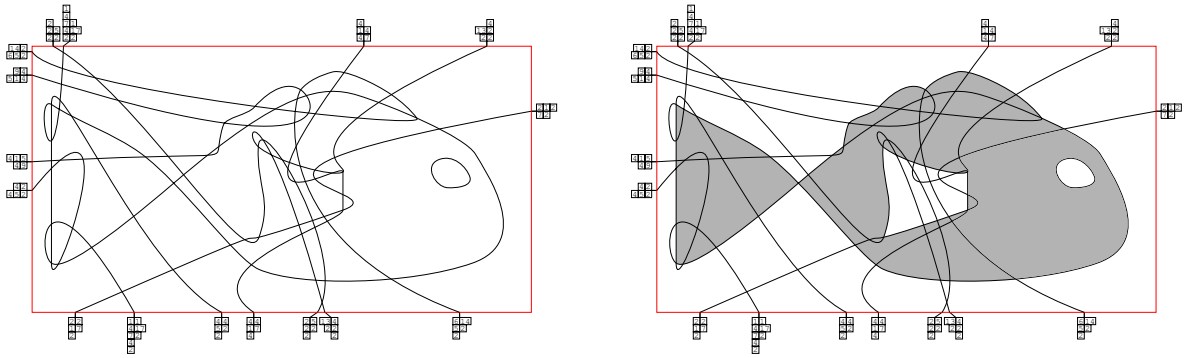


Figure 35: Output puzzle of *fish* on setting (c).

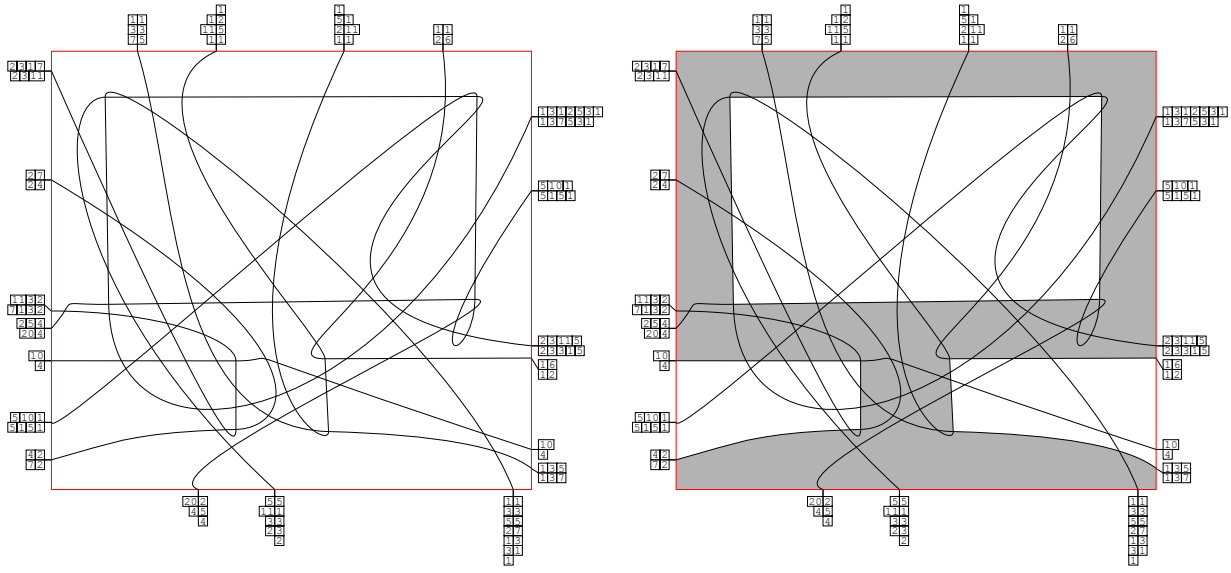


Figure 36: Output puzzle of *monitor* on setting (c).

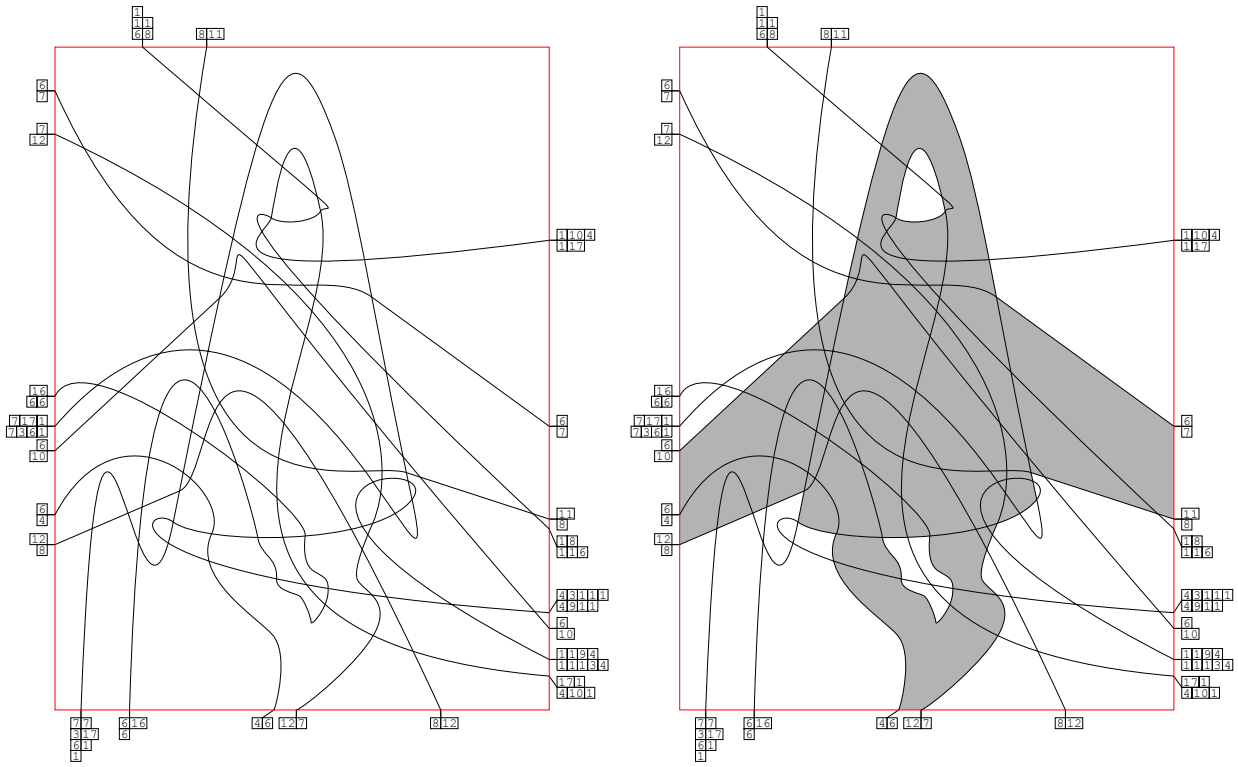


Figure 37: Output puzzle of *rocket* on setting (c).

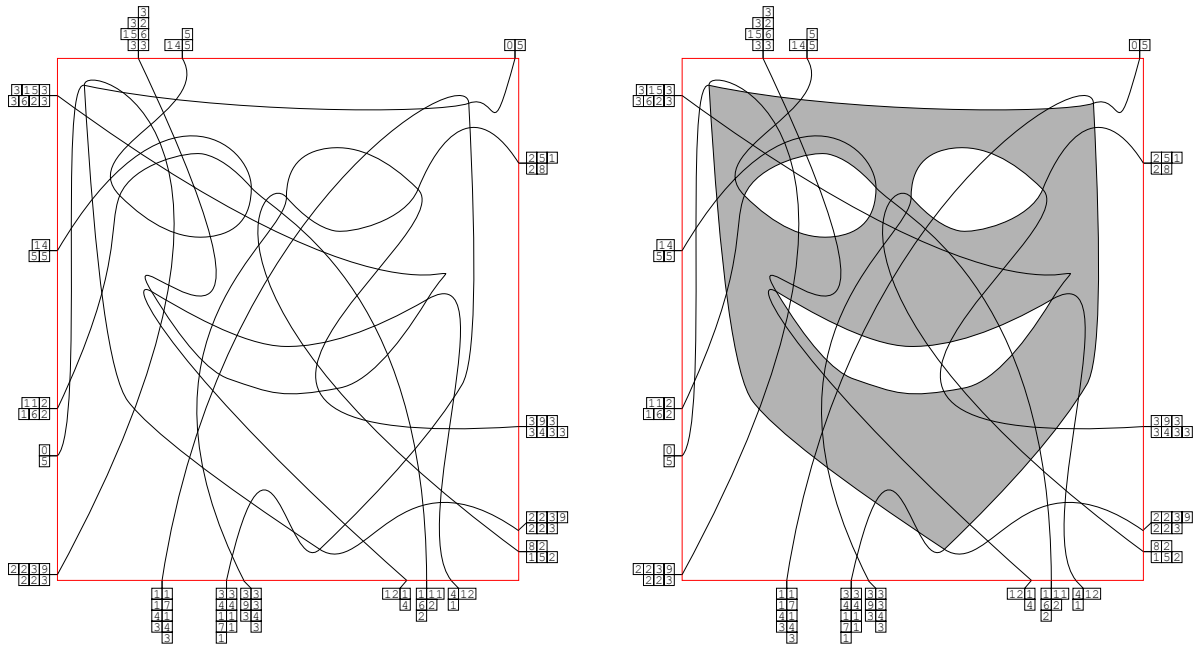


Figure 38: Output puzzle of *mask* on setting (c).

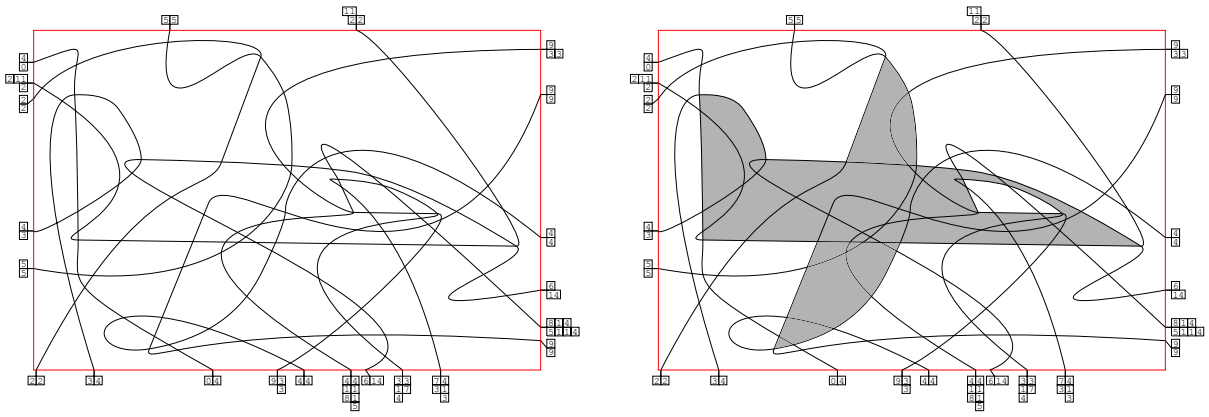


Figure 39: Output puzzle of *airplane* on setting (c).

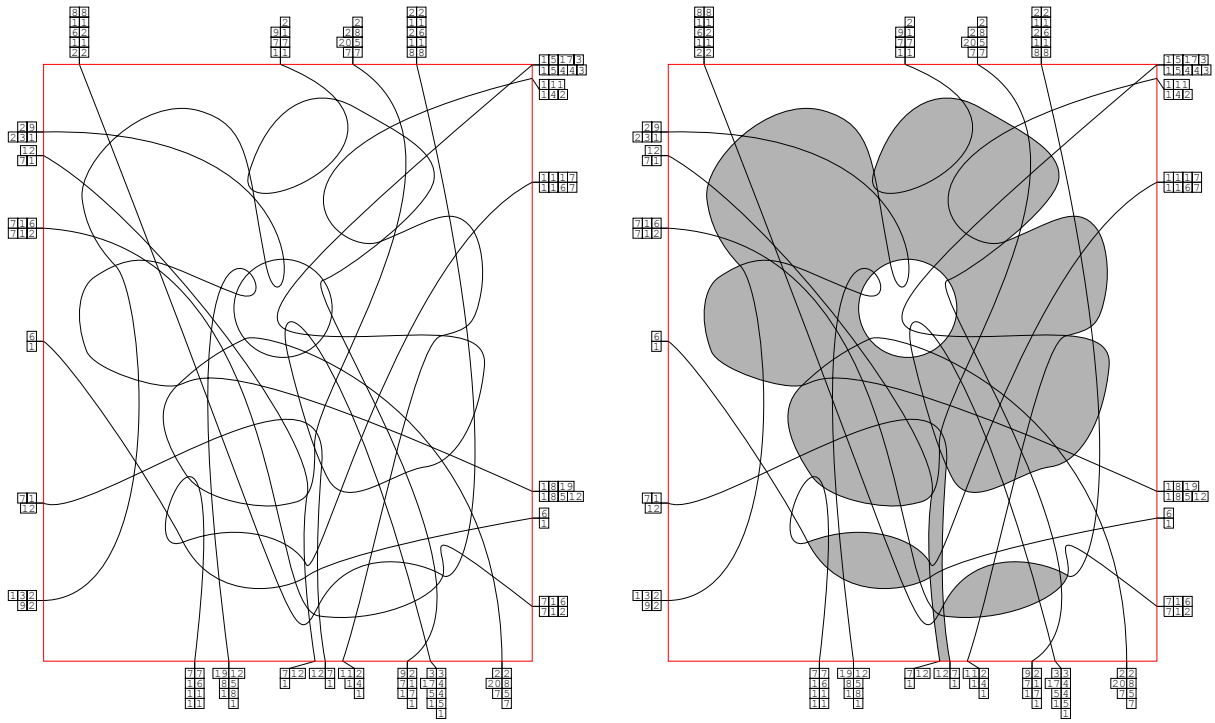


Figure 40: Output puzzle of *flower* on setting (c).

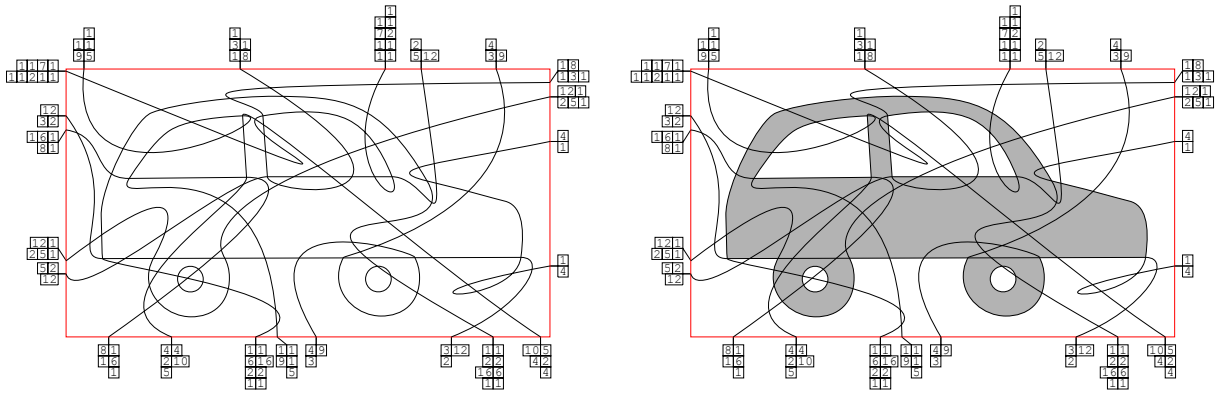


Figure 41: Output puzzle of *car* on setting (c).

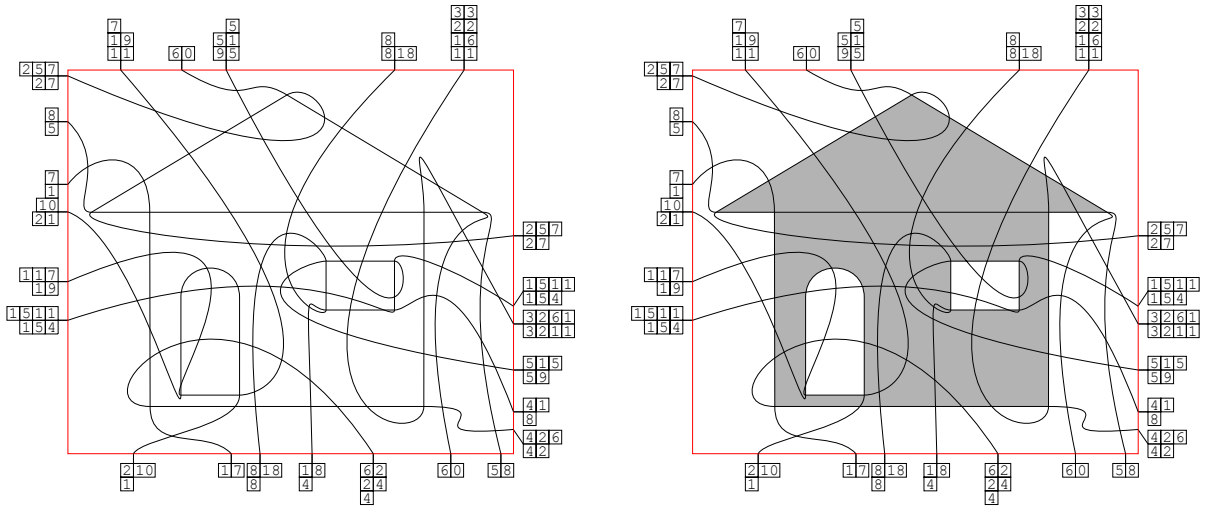


Figure 42: Output puzzle of *hut* on setting (c).

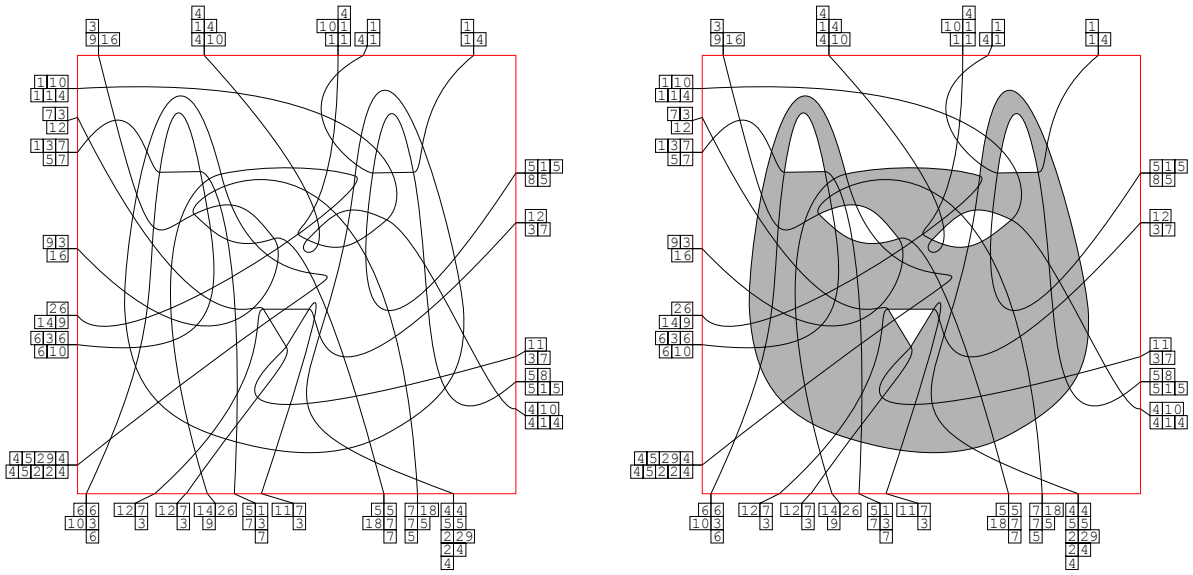


Figure 43: Output puzzle of *catface* on setting (c).

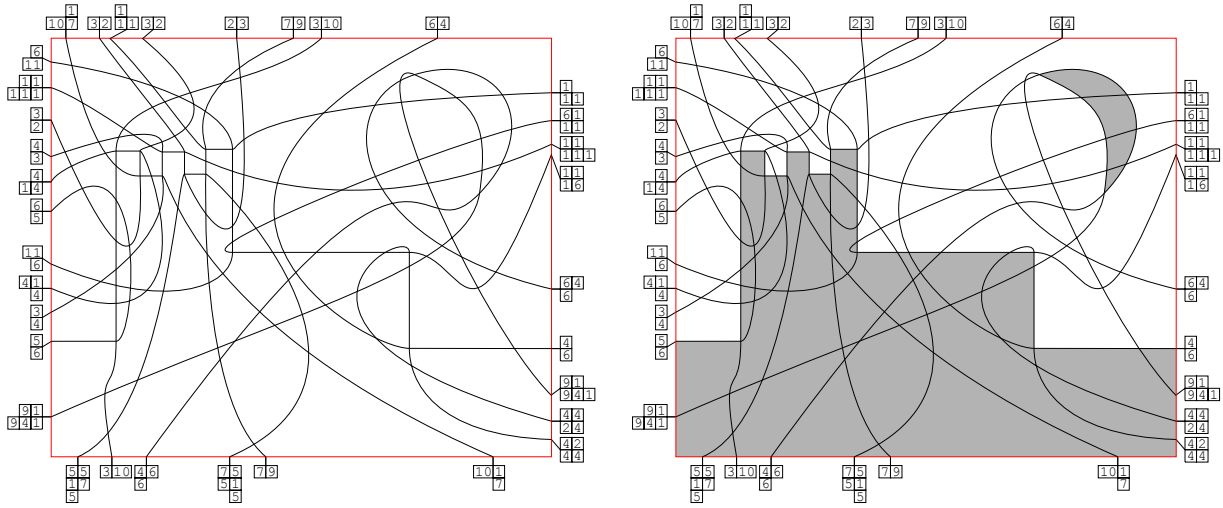


Figure 44: Output puzzle of *church* on setting (c).

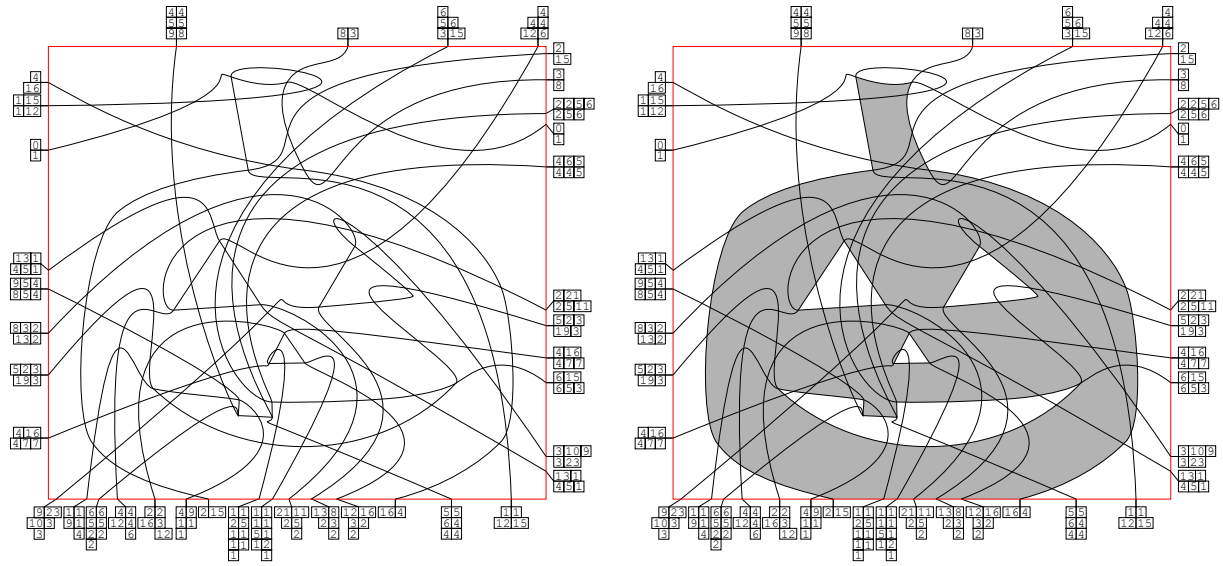


Figure 45: Output puzzle of *pumpkin* on setting (c).

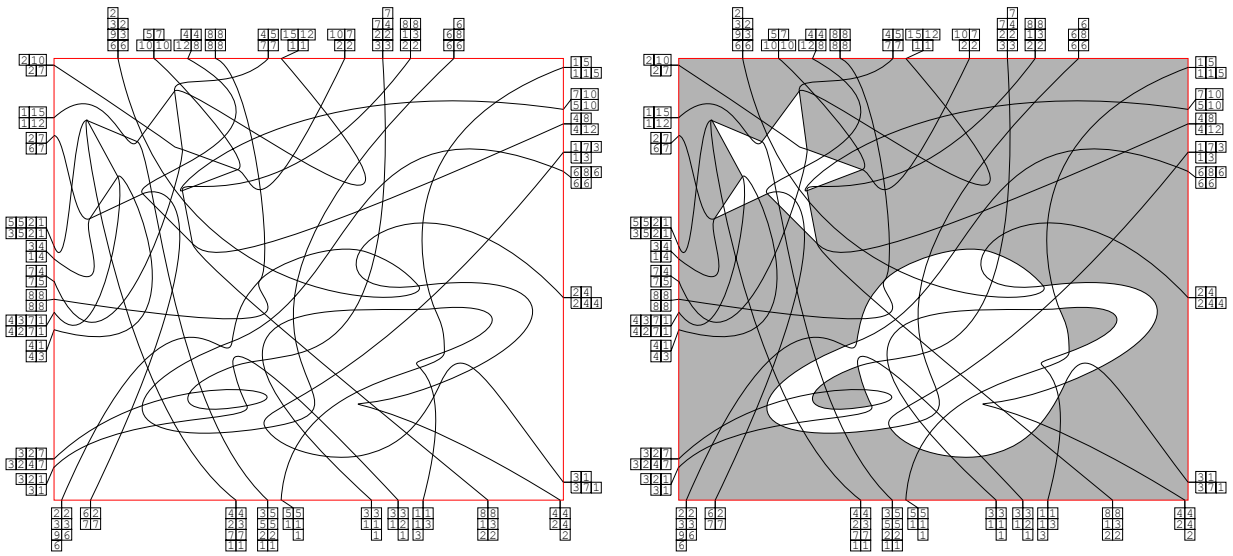


Figure 46: Output puzzle of *space* on setting (c).