Reconfiguration in Curve Arrangements to Reduce Self-Intersections and Popular Faces

Florestan Brunck
University of Copenhagen, Denmark

Hsien-Chih Chang
Dartmouth College, Hanover, NH, USA

Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Computer Science, Tulane University, New Orleans, LA, USA

Tim Ophelders
Department of Information and Computing Sciences, Utrecht University, The Netherlands
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Lena Schlipf
AI Center, Department of Computer Science, University of Tübingen, Germany

Abstract

We study reconfiguration in curve arrangements, where a subset of the crossings are marked as *switches* which have three possible states, and the goal is to set the switches such that the resulting curve arrangement has few self-intersections, or few faces that are incident to the same curve multiple times (a.k.a. *popular faces*). Our results are that these problems are NP-hard, but FPT in the number of switches. Minimizing self-intersections is also FPT in the number of *non-switchable* crossings; for minimizing popular faces this problem remains open. Our results can be applied to generating *curved nonograms*, a type of logic puzzle that has received some attention lately. Specifically, our results make it possible to efficiently convert *expert* puzzles into *advanced* puzzles (or determine that this is impossible).

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Fixed parameter tractability

Keywords and phrases Curve Arrangements, Reconfiguration, Curve Arrangements, NP-hardness, Fixed-Parameter Tractability, Puzzle Generation

Digital Object Identifier 10.4230/LIPIcs.GD.2025.36

Funding *Tim Ophelders*: Partially supported by the Dutch Research Council (NWO) under project no. VI.Veni.212.260.

1 Introduction

We study reconfiguration problems in curve arrangements in the plane. A set of open or closed curves that are well-behaved induces an *arrangement* that partitions the plane into faces, edges, and vertices; we consider a *switch* operation that locally uncrosses two crossing curves [7, 8, 10, 9, 14, 11]. Note that under this operation each arrangement vertex has three possible "states", but in two of the three states the vertex will no longer be a vertex after the switch; we therefore define *switches* to be local regions which either contain a vertex (crossing) or two disjoint strands of curves, see Figure 1.

Our terminology is formally defined in Section 1.2. We are interested in reducing the number of self-intersections and *popular faces* [5] in the arrangement using – ideally few – such switch operations.

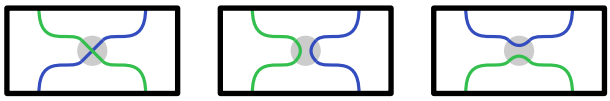

Figure 1 The three states of a switch. Only the left one contains a crossing and hence a vertex.

1.1 Motivation & Background

Our question is motivated by the problem of generating *curved nonograms*. Nonograms, also known as *Japanese puzzles*, *paint-by-numbers*, or *griddlers*, are a popular puzzle type where one is given an empty grid and a set of *clues* on which grid cells need to be colored. A clue consists of a sequence of numbers specifying the numbers of consecutive filled cells in a row or column. A solved nonogram typically results in a picture (see Figure 2(a)). There is quite some work in the literature on the difficulty of solving nonograms [1, 3, 15, 17].

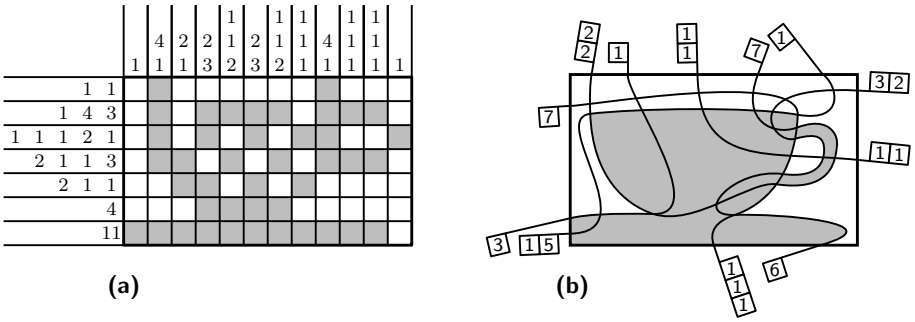

Figure 2 Two nonogram puzzles in solved states. **(a)** A classic nonogram. **(b)** A curved nonogram. (Puzzles in Figures 2 and 3 are part of the supplementary material of van de Kerkhof et al [16].)

Van de Kerkhof et al. [16] introduced *curved* nonograms, a variant in which the puzzle is no longer played on a grid but on any arrangement of curves (see Figure 2(b)). In curved nonograms, a clue specifies the numbers of filled faces of the arrangement in the sequence of faces that are incident to a common curve on one side. Van de Kerkhof et al. focused on heuristics to automatically generate such puzzles from a desired solution picture by extending curve segments to a complete curve arrangement. They observed that curved nonograms come in different flavours of increasing complexity – not in terms of how hard it is to *solve* a puzzle, but how hard it is to understand the rules (see Figure 3).

- *Basic* nonograms are puzzles in which each clue corresponds to a sequence of unique faces. The analogy with clues in classical nonograms is straightforward.
- *Advanced* nonograms may have clues that correspond to a sequence of faces in which some faces appear more than once because the face is incident to the same curve multiple times. When such a face is filled, it is also counted multiple times; in particular, it is no longer true that the sum of the numbers in a clue is equal to the total number of filled faces incident to the curve. This makes the rules harder to understand.
- *Expert* nonograms may have clues in which a single face is incident to the same curve on *both* sides. They are even more confusing than advanced nonograms.

Löffler et al. [12] strenghten this classification by observing that the different flavours of nonograms also behave differently in terms of complexity of solving - in particular, they show that the existing concept of *simple* classical nonograms [1] can be extended to curved nonograms, and that *basic* and *advanced* simple curved nonogram can still be solved in polynomial time while solving *expert* simple curved nonograms is likely to be hard.

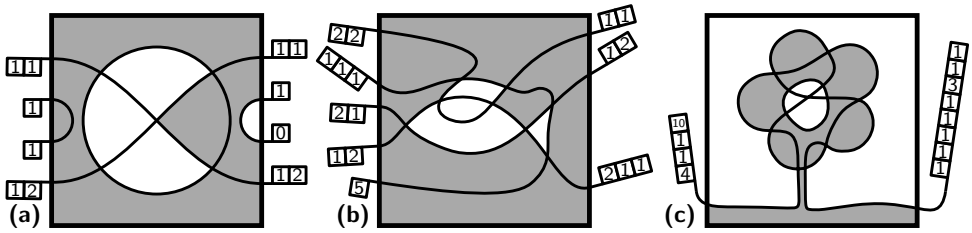

Figure 3 Three types of curved nonograms of increasing complexity [16], shown with solutions. **(a)** *Basic* puzzles have no popular faces. **(b)** *Advanced* puzzles may have popular faces, but no self-intersections. **(c)** *Expert* puzzles have self-intersecting curves. We can observe closed curves (without clues) in **(a)** and **(c)**.

Van de Kerkhof et al. stated that it would be of interest to generate puzzles of a specific complexity level. Their generators are currently not able to do so other than by trial and error. Specifically, the puzzles they generate are almost always of the "expert" complexity and occasionally of the "advanced" complexity; particularly the probability of generating large basic puzzles is so small that trial and error is not a feasible approach.

De Nooijer et al. [4, 5, 6] explore one possible way to generate nonograms of a specific complexity: rather than building a new generator from scratch, they suggest using an existing generator and modifying the output. They define a *popular face* in a curve arrangement to be a face that is incident to a single curve multiple times. With this definition, expert puzzles correspond exactly to arrangements with self-intersecting curves, and advanced puzzles correspond exactly to the presence of *popular faces* in the arrangement. Note that a self-intersection necessarily implies at least one popular face.

De Nooijer et al. investigated how to reduce the number of popular faces by *inserting* new curves into the arrangement. This approach ensures that the shapes of the faces are not distorted. But the approach also has some drawbacks: by definition it increases the size of the arrangement; also, by inserting curves we will never remove any self-intersection, so this approach is only suitable for transforming advanced puzzles into basic puzzles but not expert puzzles into advanced ones.

In this paper, we explore a different approach based on local changes to the arrangement. This will keep the size of the arrangement and its visual complexity roughly the same. However, in order to reconnect curves locally it is necessary to distort the curves a little bit, which may be undesirable for curve pieces that outline vital parts of the solution picture. Therefore, we assume that a set of *switches* is given in advance: (arbitrarily) small regions in which the arrangement may be reconfigured. Such a set of switches could be obtained either by human intervention, or automatically (for instance, one could take all crossings in the arrangement that are sufficiently far from the solution picture).

1.2 Terminology & Preliminaries

Let $\mathcal{A}$ be a set of curves which lie inside the area bounded by a closed curve F, called the *frame*. All curves in $\mathcal{A}$ are either *closed* or they are *open* with a start and end point on F. We refer to $\mathcal{A}$ as a *curve arrangement*, see Figure 4(a). We consider only *simple* arrangements, where no three curves meet in a point, and all intersections are transversal crossings (no tangencies). The arrangement $\mathcal{A}$ can be seen as an embedded multigraph whose vertices are crossings between curves and whose edges are *curve segments*. Arrangement $\mathcal{A}$ subdivides the region bounded by F into *faces*. We call a face *popular* when it is incident to multiple curve segments belonging to the same curve in $\mathcal{A}$ (see Figures 4(b–c)).

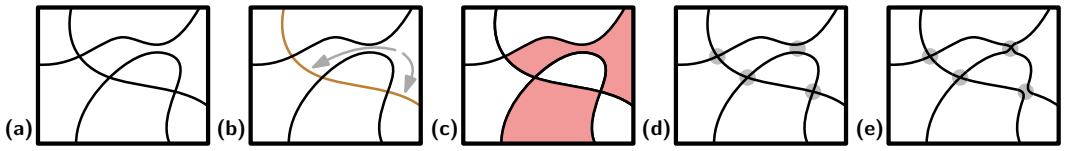

Figure 4 **(a)** An arrangement of curves inside a frame. **(b)** The highlighted curve is incident to the top right face in two segments, making the face *popular*. **(c)** All popular faces are highlighted. **(d)** A set of switches. **(e)** A possible reconfiguration after which no more faces are popular.

Now, let a *switch* be a tiny topological disk around each crossing, which is traversed by exactly two curve segments, in which we are allowed to reroute the curves of $\mathcal{A}$ (see Figures 4(d–e)). We study the problem:

▶ **Problem 1.** *Given a curve arrangement and a set of switches, can we reconfigure the curves so as to remove all, or as many as possible, self-intersections and/or popular faces? And can we minimize the number of switch operations?*

Once we are given a set of switches, the question above is essentially combinatorial. Observe that the problem is only interesting if the arrangement has at least some *non-switchable* crossings (or we want to minimize the number of switch operations).

▶ **Observation 2.** *If all intersections are switches, it is possible to remove all popular faces.*

Proof. Simply set every switch to any non-intersecting state. Then each curve bounds a single face on each side, and thus no face is popular. ◀

1.3 Results & Organization

We make the following contributions. In Section 2, we show that the problem of testing whether it is possible to remove either all self-intersections or all popular faces from an arrangement, where some vertices are marked as switchable and some as non-switchable, is NP-hard. On the other hand, when all vertices are switchable, the problem is trivial (Observation 2). However, we do show that in this case finding a solution that minimizes the number of switched vertices is also NP-hard. In Section 3, we turn our attention to parameterized results. We show that when we have an arrangement with only k switchable crossings, or k *non-switchable* crossings, the problem of removing all self-intersections is fixed-parameter tractable in k. For k switchable crossings, the same is true for removing the popular faces. Finally, we show that another natural parameter, the number of popular faces in the input, does not help: the problem remains NP-hard even when the input has a constant number of popular faces.

2 NP-hardness

In this section, we present NP-hardness results and bring a negative answer to Problem 1. We begin by examining the problem of removing all self-intersections and derive Theorem 3, using an intermediate *Permuter problem*. From there, we then reduce the problem of removing popular faces using a technique of overlaying a *global* grid. Finally, we present subsequent extensions concerning the minimal number of switch operations required by overlying *local* grids, or *waffles*. The organisation of our results in this section is summarized in Figure 5.

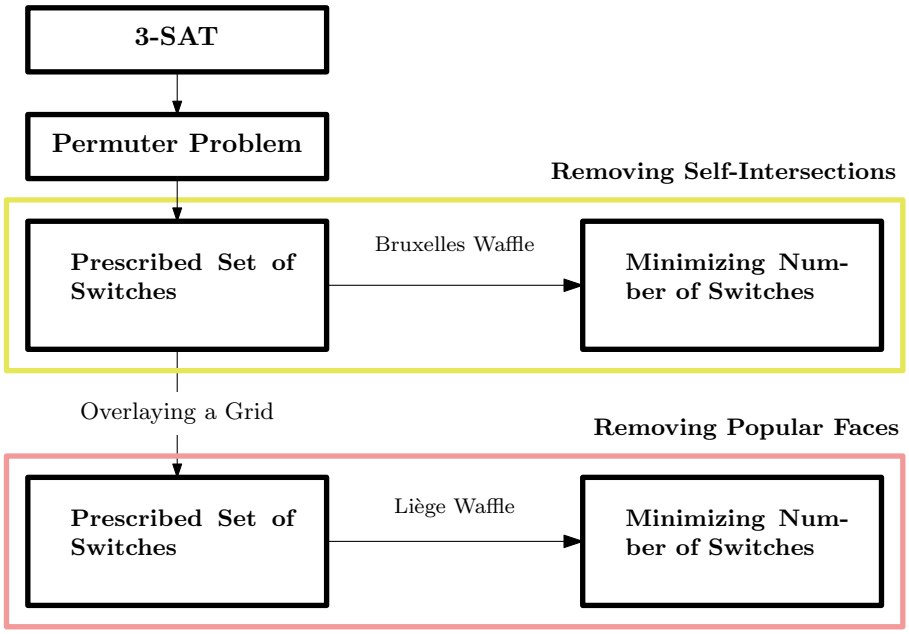

Figure 5 The flow of successive reductions underlying our NP-hardness proofs. Each arrow represents a reduction.

2.1 Base Results

We start by proving our base results: we introduce the *Permuter problem*, we show that 3-SAT reduces to the Permuter problem, and we show that the Permuter problem reduces to removing self-intersections in curve arrangements.

▶ **Theorem 3.** *Given a curve arrangement and a prescribed set of switches, it is NP-hard to decide whether it is possible to configure the switches such that the resulting arrangement has no self-intersections.*

2.1.1 The Permuter Problem and its Reduction from 3-SAT

Consider the straight-line drawing of the complete bipartite graph $K_{k,k}$ in the plane, with bi-partition I and O (referred as its *inputs* and *outputs*) such that I and O are evenly distributed on opposite sides of a rectangle R. Fixing the same linear horizontal order on the vertices of I and O, the *k-permuter* $\Pi_{k,\sigma}$ is the matching of $K_{k,k}$ associated to the permutation σ of $[n]$. An instance of the Permuter problem consists of a finite collection $\{\Pi_{k_i}, \sigma_i\}_{i\in[n]}$ of k_i-permuters, realised in the plane using an associated collection $\{R_i\}_{i\in[n]}$ of rectangles, together with a finite collection of paths $\{P_i\}_{i\in[m]}$ outside the rectangles, such that:

- Every element of $\{P_i\}_{i\in[m]}$ has both its endpoints in one of the k-permuters (possibly the same, and possibly both inputs/outputs).
- Every input and output of every permuter is connected to a unique element of $\{P_i\}_{i\in[m]}$.
- For all $i \in [m]$, for all $j \in [n]$, $P_i \cap R_j = \varnothing$; i.e., each path P_i is outside each rectangle R_j.

By construction, every path of $\{P_i\}_{i\in[m]}$ belongs to a unique closed loop. The *Permuter problem* then consists in deciding whether or not there exists a choice of n permutations $\sigma_1, \sigma_2, \ldots, \sigma_n$ of $[k_1], [k_2], \ldots, [k_n]$ such that each resulting loop is *simple* (see Fig. 6).

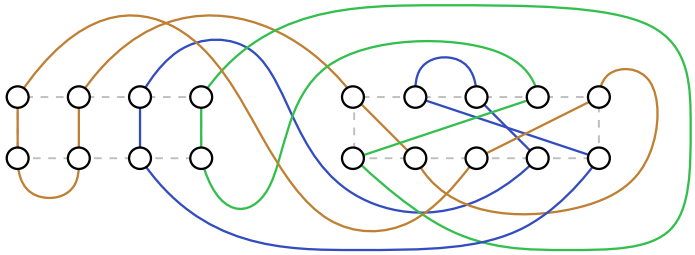

Figure 6 An instance of the k-Permuter problem with a 4-permuter and a 5-permuter, assigned with the permutations $\sigma_1 = \mathrm{Id}$ and $\sigma_2 = (12534)$. There are 3 resulting closed loops: one simple (green) and two self-intersecting (blue and brown).

We believe that the Permuter problem is of independent interest, and we show that it is NP-hard. We will prove Theorem 3 by reduction from the Permuter problem.

▶ **Theorem 4.** *The Permuter problem is NP-hard.*

Proof. The proof is by a polynomial-time reduction from 3-SAT. We shall use two types of gadgets: 2-permuters for variable assignment and 3-permuters for clause verification (see Fig. 7). For simplicity, each k-permuter is depicted by a black box on the diagram, where the value of k is made clear by the number of incoming/outgoing paths. Each different colour in the figure indicates a different variable. The thick or thin dashed lines on the top, bottom and middle-left part of the diagram indicate respectively the false and true literals of each variable. The thick and thin solid lines in the middle-right section of the diagram indicate respectively the true or false assignment of each variable. Given a Boolean formula in 3-CNF form with n variables, we construct $2n$ non-crossing semi-circular arcs. We replicate this construction twice to form the top and bottom parts of the diagram. In the middle, we show a single clause gadget, involving two 3-permuters. To simulate the two logical OR of the clause, we proceed as follows: if the corresponding 3-clause involves the variables x_i, x_j and x_k, we select the wire *opposite* to their desired truth value in the clause (i.e. thick for $\overline{x_i}$, thin for x_i) and "drag" them towards the gadget to intersect the same single path chosen among the three paths linking the output of the first 3-permuter to the inputs of the second. By construction, there is no valid permutation assignment to the two 3-permuters which avoids all possible self-intersections with the three black paths if and only if x_i, x_j and x_k all have the wrong truth assignment. Furthermore, to the right of the 3-clause gadget, we have weaved the incoming paths of the first and the outgoing paths of the second in such a way that, if the composition of the two 3-permuters were not the identity, at least one of the resulting closed loop would self-intersect. Thus each such pair of 3-permuters cannot "cheat" and has to compose to the identity. As a consequence, for each of the variables involved, the composition of its two 2-permuters must also be the identity. By construction, there are then exactly two ways of ensuring this is the case: either both 2-permuters are the identity themselves (setting the variable to be true), or both of them correspond to the transposition (12) swapping the inputs (setting the variable to be false). Fig. 8 shows the instance created for the Boolean formula $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_4} \vee x_5 \vee \overline{x_6})$. ◀

2.1.2 The Permuter Problem Reduces to the Self-Intersection Problem

▶ **Lemma 5.** *The problem of configuring a given set of switches to avoid self-intersections is polynomial-time reducible from the Permuter problem.*

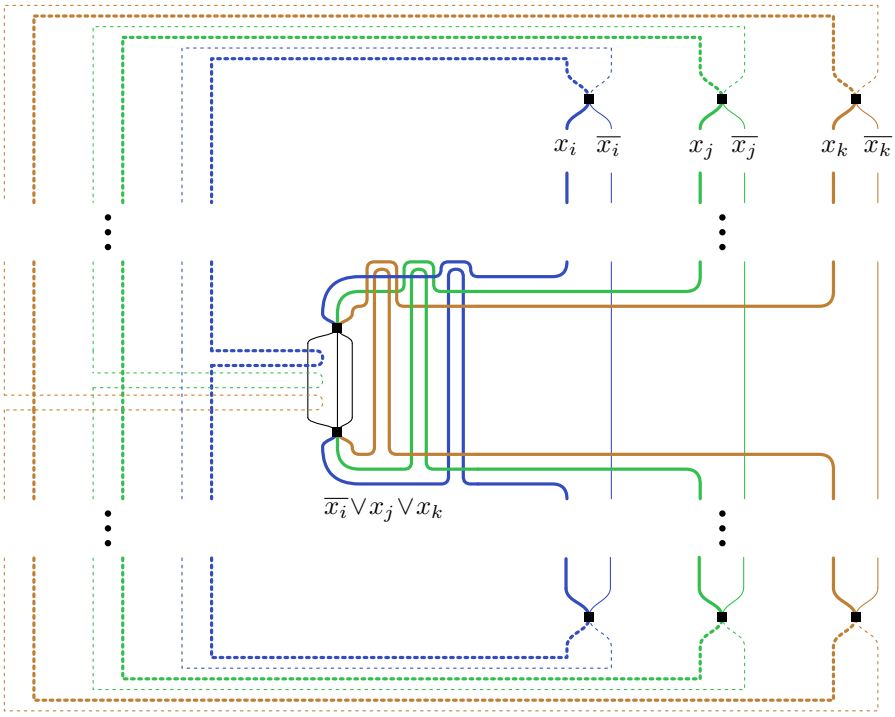

Figure 7 A single 3-clause in the reduction from 3-SAT to the Permuter problem.

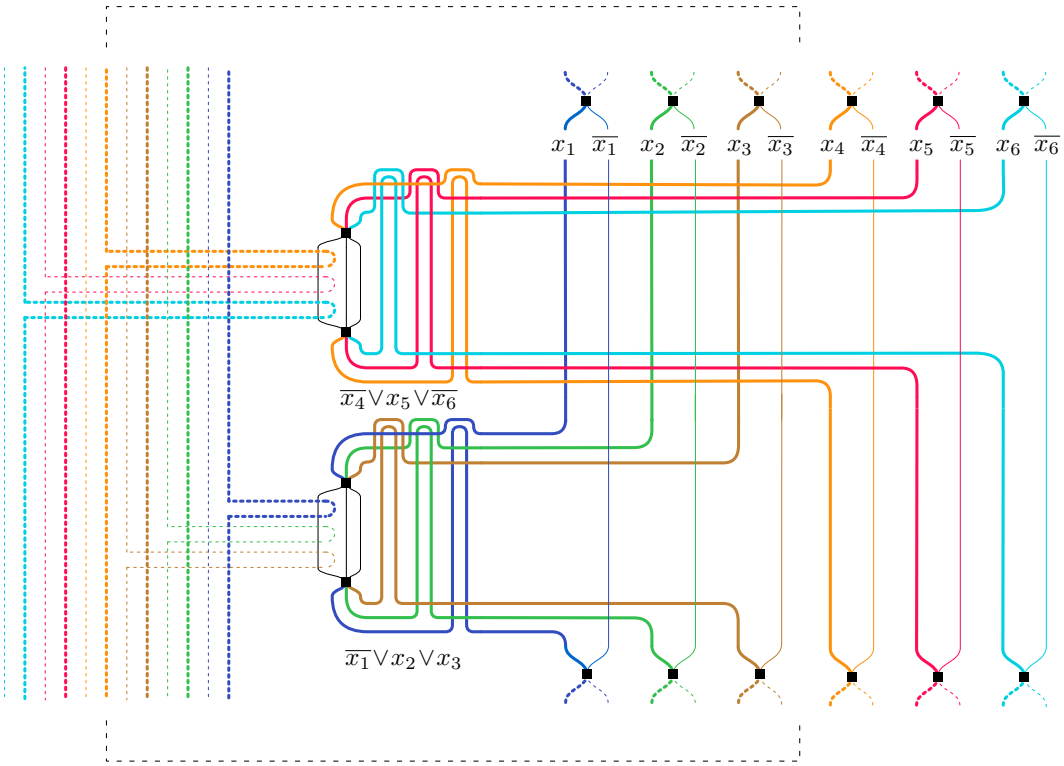

Figure 8 The instance of the Permuter problem created for the Boolean formula $(\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_4} \vee x_5 \vee \overline{x_6})$.

Proof. The proof of the claim proceeds as follows: we first construct self-intersection gadgets to simulate 2-permuters, take note of the fact that S_n is generated by transpositions and show how to use 2-permuters to construct general k-permuters. The construction for 2-permuters is presented on Fig. 9: the inputs and outputs are connected by two curves weaved into a double coil structure with two intersections, one of which is a switch. The three resulting configurations are shown on the figure; only the leftmost two are free of self-intersections.

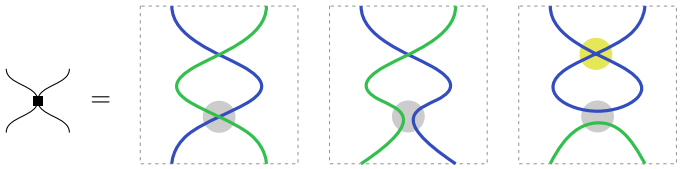

Figure 9 Constructing a 2-permuter using two "doubly-coiled" curves. The grey disk indicates the only switch available, the yellow disk highlights the self-intersection forbidding the third possibility given by the switch.

To simulate a general k-permuter, we introduce the gadget illustrated in Fig. 10. We begin with a k-by-k square and evenly distribute k inputs and outputs on its top and bottom edges, respectively. For every $i \in [n]$, the top i-th input is connected to the bottom $(n-i)$-th output by the path of slope -1 which gets reflected into a path of slope 1 upon meeting the left edge of the square. We then insert a total of $\frac{k(k-1)}{2}$ 2-permuters: one at every site where two paths intersect inside the square.

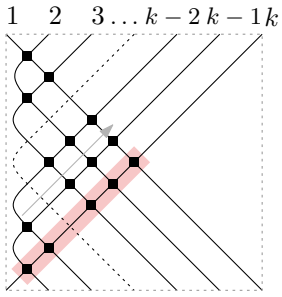

Figure 10 Constructing a k-permuter, using $\frac{k(k-1)}{2}$ 2-permuters.

While many configurations of this gadget are redundant and yield the same permutation, a short inductive argument shows that it is indeed able to simulate any permutation on k elements. The base case is simply the 2-permuter we previously described. Assume then by induction that the version of our gadget with $k-1$ inputs and outputs can successfully simulate all permutations of $[k-1]$. Note that any permutation σ of $[k]$ can be written as the composition of a permutation of $[k-1]$ followed by the insertion of the element k into one of the k positions before or after one of the $k-1$ permuted elements. It is thus enough to show that the addition of the last "row" of $k-1$ 2-permuters (highlighted in pink) can simulate this last insertion step. Labelling the $(k-1)$ 2-permuters of the $(k-1)$-th row according to the direction indicated on Fig. 10, we insert the element k in position i (before the i-th element) by setting all the 2-permuters from positions i to k to swap their inputs, and the remaining i 2-permuters to the identity. This effectively shifts all the elements with positions greater than i by 1 (the permuters reroute their corresponding wires to the segment of slope 1 instead of -1) as we sequentially shift the element k to the left i times (the permuters successively let the k-th input path "slide" on the last path of slope -1). ◀

Proof of Theorem 3. Theorem 3 now follows directly from Theorem 4 and Lemma 5. ◀

2.2 Extensions

We now extend the result from Theorem 3 in several ways, to prove that the problem remains hard when we wish to minimize the number of switch operations, or when the goal is to remove all popular faces rather than self-intersections. The idea is always to locally alter the reduction in a way that does not affect its global properties.

2.2.1 Minimizing the number of switches

In the previous section, we had a prescribed set of switches; indeed, this is necessary since if we are allowed to switch everywhere, then we can always remove all self-intersections by Observation 2.

However, now consider the scenario in which we wish to minimize the number of switch operations, or, in the decision version, we wish to test for a given k whether there exists a sequence of k switch operations such that the resulting arrangement has no self-intersections. In this scenario, we may or may not have a prescribed set of switches.

The idea is to emulate the construction from Section 2.1, but to replace every self-intersection in the construction which is *not* a switch by a *waffle gadget*. Such a gadget is built in such a way that even if *every* intersection in the gadget is a switch, the number of switches required to change its global state is more than a parameter c. If we then choose $c > k$, the result follows since essentially we are never allowed to switch these gadgets.

The Bruxelles Waffle. We introduce the *Bruxelles waffle* (in contrast to the *Liège waffle* which we describe in Section 2.2.3). The construction is illustrated in Figure 11.

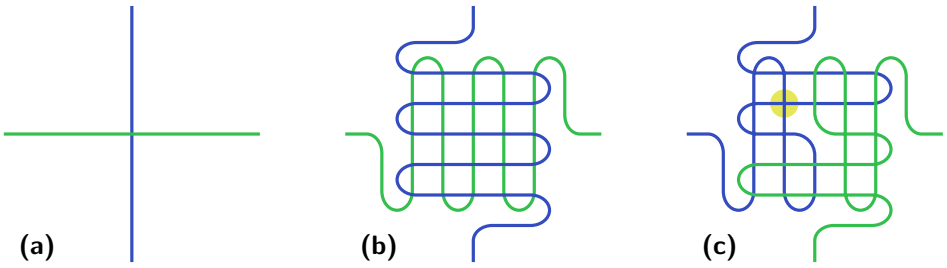

Figure 11 The Bruxelles waffle. (a) A crossing between two different curves. (b) To construct a Bruxelles waffle at the crossing, we adapt the two curves so they intersect c^2 times. (c) Any sequence of local switches to emulate a global switch must be of length at least c or otherwise lead to a self-intersection.

▶ **Lemma 6.** *In a Bruxelles waffle, any sequence of fewer than c switches must result in an arrangement with either the same combinatorial structure as the original, or at least one self-intersection.*

Proof. Assume that after fewer than c switches, no self-intersections remain. We need to show that opposite terminals lie on the same strand. Assume for a contradiction that opposite terminals do not lie on the same strand. Then the left terminal lies on the same strand as either the top or the bottom terminal. Without loss of generality (by rotational symmetry of the gadget) assume that the left terminal lies on the same strand as the bottom terminal.

Because there are c rows, at least one row has not undergone any switch, and the horizontal path in that row has a single color. Because there are no self-intersections, all strands crossing the row vertically have a color different from the horizontal path of the row.

Removing the row splits the gadget into a part above and a part below the row, and the left and right endpoints of the horizontal path connect to different such parts. The strand containing the horizontal path of the row cannot be closed up without crossing the row, so the strand connects a terminal below and a terminal above the row. That is, it connects the bottom or left terminal to the top or right terminal. This contradicts our assumption that the bottom and left terminals lie on the same strand. ◀

▶ **Theorem 7.** *Given a curve arrangement and an integer k, it is NP-hard to decide whether there exists a sequence of k switches such that the resulting arrangement has no self-intersections.*

2.2.2 Removing popular faces

Next, we consider the problem of removing all popular faces rather than only self-intersections.

The idea is to globally overlay the construction from Section 2.1 with a sufficiently fine grid of horizontal and vertical lines, in which none of the intersections with these new lines are switches. In particular, we make the grid such that each cell of the grid contains either a single strand of one of the curves, or two strands that intersect, or nothing at all (see Figure 12).

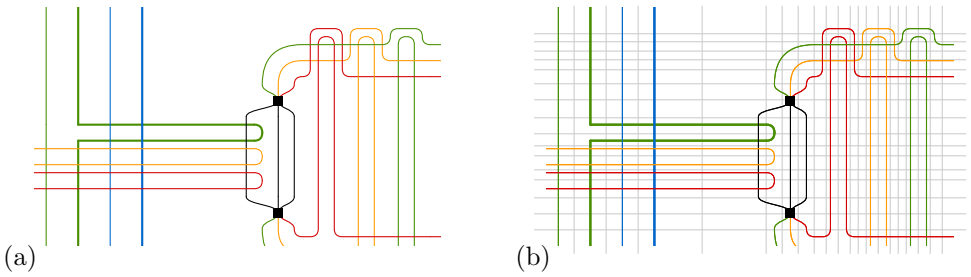

Figure 12 (a) A small section of the construction. (b) A fine grid that separates all elements of the construction into separate cells.

After we overlay the grid, all (new) faces are incident to each grid line only once, and incident to each original curve only once, unless two crossing strands inside a grid cell belong to the same curve. In other words, all remaining popular faces are now due to self-intersections. Therefore, in order to remove all popular faces we need to remove all self-intersections, and this is also sufficient.

▶ **Theorem 8.** *Given a curve arrangement and a prescribed set of switches, it is NP-hard to decide whether it is possible to configure the switches such that the resulting arrangement has no popular faces.*

2.2.3 Removing popular faces with a minimum number of switches

Finally, we consider the setting where every intersection is an allowed switch; in this case, testing whether all popular faces can be removed is again not hard by Observation 2. However, when we wish to remove all popular faces using a minimum number of switch operations, the problem remains NP-hard. The idea is similar to that in Section 2.2.1, but we will need to use a different gadget that ensures we cannot perform any switches (since having no self-intersections does not necessarily imply there are no popular faces).

The Liège Waffle. We introduce the *Liège waffle*, illustrated in Figure 13.

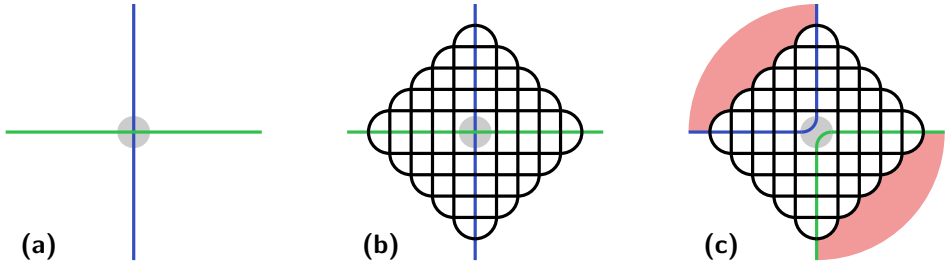

Figure 13 The Liège waffle. (a) A crossing between two different curves. (b) To construct a Liège waffle, we insert a set of c new closed curves that surround the crossing. (c) Any sequence of local switches to emulate a global switch must be of length at least c or lead to a popular face.

Essentially, we overlay c new closed curves on the two (crossing) terminal strands, such that each of the new curves is incident to each of the four unbounded faces, and that the disk bounded by the curve contains the crossing between the two terminal strands.

▶ **Lemma 9.** *In a Liège waffle, any sequence of fewer than c switches must result in an arrangement with either the same combinatorial structure as the original, or at least one popular face.*

Proof. If we change the global connectivity, then one of the four unbounded faces will globally have two strands of the same curve; say the top left face (Figure 13 (c)). In order for this face to *not* be popular, these two strands must be consecutive along the curve. Initially, all intersections incident to the face are crossing, so they must all be uncrossed. There are $c+1$ such intersections, so we need at least $c+1$ switches. ◀

▶ **Theorem 10.** *Given a curve arrangement and an integer k, it is NP-hard to decide whether there exists a sequence of k switches such that the resulting arrangement has no popular faces.*

3 Parameterized Results

Since the problem in general is NP-hard, we now turn our attention to parameterised complexity. We can identify several natural parameters:

- the number of popular faces in a provided configuration; or
- the number of switches; or
- the number of non-switchable crossings.

The number of popular faces is perhaps the most natural parameter when trying to remove popular faces; in particular, existing heuristics for generating curve arrangements do attempt to minimize the number of popular faces and often succeed in reducing their number, but usually not all [16]. Unfortunately, as we show in Section 3.1, this does not make the problem easier. Instead, we can also consider the number of switches as a parameter. When there is only a small number of switches, it is easy to see that the problem becomes polynomial, see Section 3.2. However, in our nonogram generation application, it seems more reasonable to assume the number of *non-switchable* crossings is small: we may wish to maintain some key features of the solution image, but otherwise want to have as much freedom as possible. Here, we prove the problem is also in FPT when the goal is to reduce the number of self-intersections; for removing popular faces, the problem remains open. In Section 3.3, we explore this parameter in detail. Our parameterized results are summarized in Table 1.

Table 1 Results on parameterized complexity.

parameter	avoid self-intersections	avoid popular faces
number of popular faces	NP-hard	NP-hard
number of switchable crossings	FPT	FPT
number of non-switchable crossings	FPT	open

3.1 Parameter: number of faces

We show that the problem of removing all popular faces remains hard if we are given a starting configuration with only a constant number of popular faces. For this, we set the permuters in the construction in Section 2.1.2 to some an initial state that leads to only few popular faces but does not "help" with finding the actual solution. In particular, an assignment of a 3-SAT formula that satisfies all but one clause corresponds to a setting of permuters that results in constantly many popular faces.

We first show that 3-SAT remains hard if the input comes with an assignment that violates at most one clause; call this problem Almost-3-SAT.

▶ **Lemma 11.** *Almost-3-SAT is NP-complete.*

Proof. NP-membership is trivial. To show NP-hardness, assume without loss of generality that P≠NP, and that we have a polynomial time algorithm Alg for deciding Almost-3-SAT. We can use Alg to also output a satisfying assignment if the instance is satisfiable (by guessing the value of a variable, removing all clauses satisfied by that variable, and calling Alg on the resulting formula). We hence assume that Alg also outputs a satisfying assignment (if one exists).

We show that we can use Alg to solve 3-SAT. Consider an instance F of 3-SAT that consists of m clauses. We can solve F by invoking Alg m times, namely on subformulas of F. Suppose that we have an assignment A that satisfies the first k clauses of F. Then the first $k+1$ clauses of F together with A form an instance of Almost-3-SAT, which we can solve using Alg to either obtain an assignment A' that satisfies the first $k+1$ clauses of F, or a proof that the first $k+1$ clauses of F are not satisfiable, and hence F is not satisfiable. Starting with $k=0$, and repeating the above m times and incrementing k, we either obtain a satisfying assignment for F, or discover that F is not satisfiable. Hence, Alg can be used to solve 3-SAT in polynomial time. ◀

▶ **Theorem 12.** *Given a curve arrangement with a constant number of popular faces, the problem of deciding whether there exists a configuration without any popular faces (or self-intersections) is still NP-complete.*

Proof. Setting the permuters to an assignment that satisfies all except one clause results in configuration in which all popular faces are incident to the clause gadget that corresponds to the unsatisfied clause. Because a clause gadget has constant complexity, the configuration has a constant number of popular faces. The popular faces can be removed by reconfiguring switches if and only if the instance of Almost-3-SAT is satisfiable. ◀

3.2 Parameter: number of switches

Next, we show that the problem is not NP-hard when the number of switches is small.

▶ **Observation 13.** *Given a curve arrangement with k switches, the problem of deciding whether there exists a configuration without any popular faces (or self-intersections) is FPT in k.*

Proof. We can simply try all 3^k combinations and then check whether the resulting arrangement has self-intersections and/or popular faces. Both of these can be done in time proportional to the complexity of the arrangement. So, the running time of this algorithm is $O(3^k \cdot n)$. ◀

3.3 Parameter: number of non-switchable crossings

Finally, we explore our third parameter: the number of *non-switchable* crossings in the input arrangement. Throughout this section, we will denote this parameter as k. The case $k = 0$ corresponds to Observation 2 and is easy. We will first explore the situation when $k = 1$.

3.3.1 Removing self-intersections with exactly 1 non-switchable crossing

This question is already interesting when there is just *one* crossing that cannot be switched. As a simple example, consider the arrangement in Figure 14, which shows an arrangement with five switches, one of which is marked as non-switchable. There are exactly three states in which there are no self-intersections, but to get from one to another, four switch operations need to be done; that is, every single switchable switch needs to be flipped.

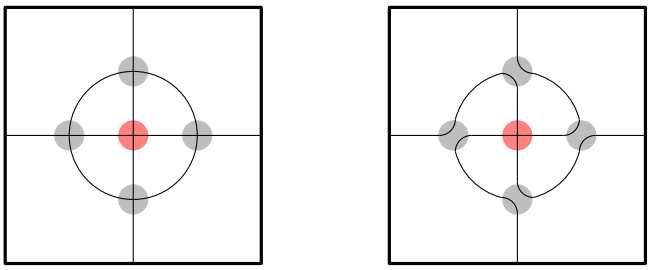

Figure 14 An arrangement with five switches, the middle of which is non-switchable. Left: one possible state without self-intersections. Right: another state without self-intersections. A third state is the mirror image of the right figure.

Define an Eulerian partition of a multigraph to be set of cycles and paths connecting boundary edges, such that each edge is used exactly once. That is, each edge lies on a single path or cycle, and no path or cycle revisits an edge. We say that an Eulerian partition is simple if each of its paths and cycles is simple. For a surface-embedded graph, we endow each vertex (of even degree) with the involution that pairs up incident edges on "opposite" sides. An Eulerian partition is straight at a vertex v if whenever a path or cycle passes through v, it does so using the edges related to each other by the involution.

Let G be the graph of the arrangement of the curves, where each switch and non-switch corresponds to a vertex of G, and every maximal segment of the curves becomes an edge of G. Assume that G has a single (degree 4) vertex v that is a non-switchable crossing. We are looking for a simple Eulerian partition of G that is straight at v. Let $G - v$ be the graph obtained by subdividing the four edges incident to v using (in cyclic order) vertices $v_1, \ldots, v_4$, and then removing v. We call v a cut-vertex if $G - v$ has more components than G.

▶ **Lemma 14.** *For any graph G that comes from a curve arrangement, and any non-boundary vertex v of G, exactly one of the following is true:*

1. *There exists a simple Eulerian partition of G that is straight at v; or*
2. *v is a cut-vertex and at least one component of $G - v$ incident to v_i $(i = 1, \ldots, 4)$ does not touch the boundary.*

Proof. We first show that Item 2 rules out Item 1. Let C be a component of $G - v$ that does not touch the boundary and without loss of generality assume that it contains v_1 (otherwise relabel the v_i's). Because C does not touch the boundary, the vertices of C of degree 1 form a subset of $\{v_1, v_2, v_3, v_4\}$. Because v is a cut-vertex of G, there exists some v_i, $i \in \{2, 3, 4\}$ that does not lie in C. By the handshaking lemma, C contains exactly two vertices of degree 1.

We claim that C *does not contain* v_3. Suppose for a contradiction that C contains v_3, then the path from v_1 to v_3 corresponds to a cycle c through v in G that by the Jordan curve theorem separates the components C_2 and C_4 containing v_2 and v_4 respectively, so $C_2 \neq C_4$. By the handshaking lemma, C_2 and C_4 must have an odd (and therefore nonzero) number of boundary ports (to compensate for the vertex v_2 or v_4 of degree 1). However, C_2 or C_4 lies interior to the disk bounded by c and hence has no boundary ports, which is a contradiction, so C does not contain v_3.

Assume without loss of generality that C contains v_1 and v_2 (the argument for v_1 and v_4 is symmetric). Now, if the Eulerian partition of G that was straight at v is simple, then $G - v$ must contain an Eulerian partition that does not contain a path connecting v_1 to v_2 (otherwise the path would self-intersect at v in G). However, because these are the only vertices of C of degree 1, and Eulerian partition of $G - v$ contains such a path, so Item 2 rules out Item 1.

It remains to construct a simple Eulerian partition (that is straight at v) if v is not a cut-vertex, or if all components of $G - v$ incident to v_i ($i \in \{1, \ldots, 4\}$) touch the boundary. Let C_i ($i = 1, \ldots, 4$) be the component of $G - v$ containing v_i. We first show that there exists a simple Eulerian partition that is straight at v if $C_1 = C_3$ (and by symmetric argument if $C_2 = C_4$). Assume that $C_1 = C_3$, then $G - v$ contains a simple path π_1 from v_1 to v_3. Let $G - v - \pi_1$ be the graph obtained from $G - v$ by removing the edges of π_1. If v_2 and v_4 are connected in $G - v - \pi_1$, then there exists a simple path π_2 from v_2 to v_4. Then π_1 and π_2 correspond to simple cycles c_1 and c_2 in G that are straight at v, and the remainder of G admits a Eulerian partition (and hence a simple one). If instead v_2 and v_4 are not connected after removing the edges of π_1 from $G - v$, then we pick an arbitrary Eulerian partition of $G - v - \pi_1$. This partition corresponds to a partition of G in which π_1 corresponds to a simple cycle that is straight at v, and the paths that contain v_2 and v_4 (call them π_2 and π_4) do not intersect because they lie in different components of $G - v - \pi_1$. Hence, the paths π_2 and π_4 can be concatenated by replacing the end points v_2 and v_4 by the vertex v in G, to obtain a simple path in G that is straight at v.

We have established that there exists a simple Eulerian partition that is straight at v if $C_1 = C_3$ or $C_2 = C_4$, so assume that $C_1 \neq C_3$ and $C_2 \neq C_4$. Then v is a cut-vertex, so it remains to show that if each C_i touches the boundary, then there exists a simple Eulerian partition that is straight at v. We claim that $G - v$ contains edge-disjoint simple paths π_i ($i = 1, \ldots, 4$) that connect each v_i to the boundary. For this, let π_1 and π_3 be arbitrary simple paths and consider the component C_2' of $G - v - \pi_1 - \pi_3$ that contains v_2. We claim that this component contains a boundary vertex. Indeed, by the handshaking lemma, C_2' contains an even number of vertices of degree 1, and one of them is v_2. None of the vertices v_1, v_3, v_4 lie in C_2', because v_1 and v_3 were isolated by removing the edges of π_1 and π_3, and v_4 lies on a different component because $C_2 \neq C_4$. Hence, C_2' contains a boundary vertex, and hence a simple path π_2 to it from v_2. By a symmetric argument, a component of $G - v - \pi_1 - \pi_3$ contains a simple path π_4 from v_4 to a boundary vertex. Because $C_1 \neq C_3$ and $C_2 \neq C_4$, the "concatenations" of π_1 with π_3 and of π_2 with π_4 in G are simple and straight at v. The remainder of G (after removing these two paths) admits a Eulerian partition and hence a simple one. Reinserting the two paths yields a simple Eulerian partition of G that is straight at v. ◀

This now leads to the following result:

▶ **Theorem 15.** *For an instance with exactly 1 non-switchable crossing, we can decide whether there exists a configuration without self-intersections in $O(n^2)$ time.*

Proof. Suppose we have an arrangement $\mathcal{A}$ for which the answer is yes, and let G be its (planar) graph. G has one special vertex ξ that represents the non-switchable crossing. We augment G by adding a new vertex ω that represents the outer face; we create an edge from all vertices on the frame to ω Now consider a configuration K for $\mathcal{A}$ without self-intersections. All curves in $\mathcal{A}$ are now edge-disjoint cycles in G. Now, consider the cycles that pass through ξ. Note that these must be two cycles; we will refer to them as the *red* cycle and *blue* cycle.

Now, replace ξ by four degree 1 vertices that we refer to as *terminals*. Let G' be the resulting (planar) graph. Note that now, we have a red and a blue *path* in G', and the paths incident to opposing terminals have the same color.

Now, our goal is to test whether such a configuration K can exist, based only on the information in G. That is, we need to find whether there exist two edge-disjoint paths in G' that connect opposite pairs of terminals. The existence of edge-disjoint paths that connect two pairs of terminals can be tested in $O(n^2)$ time [13].

If so, by Lemma 14, we then have a Eulerian partition of the remainder, and we can cover it by edge-disjoint cycles. In addition, when we re-insert ξ, the two paths become an edge-disjoint red and blue cycle that are straight at ξ. ◀

The proof of Theorem 15 is constructive, and leads to the following algorithm:

▶ **Algorithm 16.**

- *Construct G' by connecting all boundary vertices to an additional vertex ω on the outer face, and replacing ξ by four terminals.*
- *Assign alternating colors to the terminals.*
- *Run [13] (in $O(n^2)$ time).*
- *If it returns false, return false. Otherwise:*
- *For the two paths found by [13], set all switches along the paths accordingly so that two curves will follow exactly these paths.*
- *For the remainder, (greedily) cover G' with edge-disjoint cycles. If any cycle has a self-intersections, simply split it into multiple cycles.*
- *Set all remaining switches so that each cycle corresponds to a curve.*

3.3.2 Removing self-intersections with k non-switchable crossings

Next, we can extend the ideas from the previous section to k non-switchable crossings.

▶ **Theorem 17.** *For an instance with k non-switchable crossings, we can decide whether there is a configuration without self-intersections in $O(k^{k^{k^{k^{\cdots^{k}}}}} \cdot n^2)$ time.*

Proof. Suppose we have an arrangement $\mathcal{A}$ for which the answer is yes, and let G be its (planar) graph. We augment G by adding a new vertex ω that represents the outer face; we create an edge from all vertices on the frame to ω Now consider a configuration K for $\mathcal{A}$ without self-intersections. All curves in $\mathcal{A}$ are now edge-disjoint cycles in G.

Consider the cycles that pass through at least one non-switchable crossing. Because each non-switchable crossing involves only two cycles, there are at most $2k$ such cycles, and we will assign distinct colors to these cycles.

Replace each non-switchable crossing by four terminals corresponding to the four incident edges. Let G' be the resulting graph. This cuts up the cycles that pass through the non-switchable crossings into exactly $2k$ edge-disjoint colored paths that each start and end at terminals. For each non-switchable crossing, the paths incident to opposing terminals have the same color. On the other hand, because any cycle passes through any non-switchable crossing only once, every non-switchable crossing is incident to two distinct colors, so any two of its terminals that are not opposing have different colors.

We wish to recover a configuration K' without self-intersections for G based on only the following information derived from K:

1. the pairs of terminals that are connected by the $2k$ edge-disjoint paths; and
2. for each such pair, the color of the path that connects them.

For a given graph, one can decide the existence of edge-disjoint paths that connect specified pairs of terminals (and report the paths if they exist) in $O(k^{k^{k^{k^{\cdots^{k}}}}} \cdot n^2)$ time [13]. If we apply this algorithm to G' with the above information derived from K, it will return that there exists a set of edge-disjoint paths.

For any such set of edge-disjoint paths, we can correspondingly color the edges of G, and give all the edges that do not lie on any of the $2k$ paths a new color (e.g. color $2k+1$). This results in an Eulerian partition of G that is straight at each of the non-switchable crossings. Any switch will either be incident to four edges of the same color, or two pairs of two edges with the same color. We now define our configuration K' as follows. If the four edges incident to a switch s have the same color, we configure s so as not to create a crossing. If on the other hand, s is incident to two pairs of edges with different colors, we configure s in the obvious way: connecting both pairs of edges with the same color. Now, each strand of K' has a single color. We show that no strand of K' intersects itself. Indeed, any self-intersection must occur at a vertex incident to four edges of the same color. Because non-switchables are not incident to four edges of the same color, the only vertices at which a strand might self-intersect is at a switch. However, by construction, switches whose four incident edges all have the same color are configured not to cross. Hence, configuration K' has no self-intersections. ◀

Again, the proof is constructive; we now have all the ingredients for our main algorithm:

▶ **Algorithm 18.**

- *Construct G' by connecting all boundary vertices to an additional vertex ω on the outer face, and replacing each non-switchable crossing by four terminals.*
- *Call a coloring of the terminals* valid *if it uses at most $2k$ colors, and for each non-switchable crossing, the opposing terminals have the same color, and non-opposing terminals have distinct colors. There are at most $k^{2k(2k-1)}$ valid colorings.*
- *For each valid coloring of the terminals:*
  - *For a given coloring, call a matching between the terminals valid if it matches only pairs of terminals that have the same color. For a given coloring, there are at most $(2k-1)!!$ valid matchings [2].*
  - *For each valid matching:*
    - *Run [13] (in $O(k^{k^{k^{k^{\cdots^{k}}}}} \cdot n^2)$ time).*
    - *If it returns true, store the resulting collection of paths.*
- *If no paths were stored, return false.*
- *Otherwise, set all switches along the stored paths accordingly so that they become curves.*

- *For the remainder, (greedily) cover G' with edge-disjoint cycles. If any cycle has a self-intersections, simply split it into multiple cycles.*
- *Set all remaining switches so that each cycle corresponds to a curve.*

The above ideas can be partially extended to the case of removing popular faces. However, the general question whether removing popular faces is FPT in the number of non-switchable crossings remains open.

4 Conclusion

We have shown that the problem of minimizing either the number of self-intersections or the number of popular faces in a curve arrangement is NP-hard. However, in our application of puzzle generation, we expect the number of non-switchable crossings to be small, and we show that minimizing the number of self-intersections is FPT in the number of non-switchable crossings. This makes our results potentially suitable for generating advanced, rather than expert, curved nonograms, although one remaining question from a practical point of view is whether the dependency on k can be improved. On the other hand, the problem of minimizing the number of popular faces remains open, which means we cannot yet (efficiently) generate basic curved nonograms using this method.

References

1 Kees Batenburg and Walter Kosters. On the difficulty of nonograms. *ICGA journal*, 35:195–205, December 2012. doi:10.3233/ICG-2012-35402.

2 David Callan. A combinatorial survey of identities for the double factorial. https://arxiv.org/abs/0906.1317, 2009.

3 Yen-Chi Chen and Shun-Shii Lin. A fast nonogram solver that won the TAAI 2017 and ICGA 2018 tournaments. *ICGA Journal*, 41(1):2–14, 2019. doi:10.3233/ICG-190097.

4 Phoebe de Nooijer. Resolving popular faces in curve arrangements. Master's thesis, Utrecht University, 2022. URL: https://studenttheses.uu.nl/handle/20.500.12932/494.

5 Phoebe de Nooijer, Soeren Nickel, Alexandra Weinberger, Zuzana Masárová, Tamara Mchedlidze, Maarten Löffler, and Günter Rote. Removing popular faces in curve arrangements. In *Proc. 31st International Symposium on Graph Drawing*, 2023.

6 Phoebe de Nooijer, Soeren Terziadis, Alexandra Weinberger, Zuzana Masárová, Tamara Mchedlidze, Maarten Löffler, and Günter Rote. Removing popular faces in curve arrangements. *Journal of Graph Algorithms and Applications*, 28(2):47–82, November 2024. doi:10.7155/jgaa.v28i2.2988.

7 Cole A. Giller. A family of links and the Conway calculus. *Transactions of the American Mathematical Society*, 270(1):75–109, 1982.

8 Jim Hoste. The Arf invariant of a totally proper link. *Topology and its Applications*, 18(2–3):163–177, 1984.

9 Vaughan F. R. Jones. On knot invariants related to some statistical mechanical models. *Pacific journal of mathematics*, 137(2):311–334, 1989.

10 Louis H. Kauffman. State models and the Jones polynomial. *Topology*, 26(3):395–407, 1987.

11 Louis H. Kauffman. Gauss codes, quantum groups and ribbon Hopf algebras. *Reviews in Mathematical Physics*, 5(4):735–773, 1993.

12 Maarten Löffler, Günter Rote, Soeren Terziadis, and Alexandra Weinberger. On solving simple curved nonograms. In *Proc. 36th International Workshop on Combinatorial Algorithms (IWOCA 2025)*, 2025.

13 Neil Robertson and Paul D. Seymour. Disjoint paths—a survey. *SIAM Journal on Algebraic Discrete Methods*, 6(2):300–305, 1985. doi:10.1137/0606030.

14 Alexander Schrijver. On the uniqueness of kernels. *Journal of Combinatorial Theory, Series B*, 55(1):146–160, May 1992. doi:10.1016/0095-8956(92)90038-Y.

15 Nobuhisa Ueda and Tadaaki Nagao. NP-completeness results for NONOGRAM via parsimonious reductions. Technical Report TR96-0008, Department of Computer Science, Tokyo Institute of Technology, 1996. CiteSeerX 10.1.1.57.5277, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5277.

16 Mees van de Kerkhof, Tim de Jong, Raphael Parment, Maarten Löffler, Amir Vaxman, and Marc J. van Kreveld. Design and automated generation of Japanese picture puzzles. *Comput. Graph. Forum*, 38(2):343–353, 2019. doi:10.1111/cgf.13642.

17 Jan van Rijn. Playing games: The complexity of Klondike, Mahjong, nonograms and animal chess. Master's thesis, Leiden Institute of Advanced Computer Science, Leiden University, 2012.