Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 28, no. 2, pp. 47–82 (2024)
DOI: 10.7155/jgaa.v28i2.2988

Removing Popular Faces in Curve Arrangements

Phoebe de Nooijer[1] *Soeren Terziadis*[2] *Alexandra Weinberger*[3] *Zuzana Masárová*[4]
Tamara Mchedlidze[1] *Maarten Löffler*[1] *Günter Rote*[5]

[1]Geometric Computing Group, Utrecht University, Utrecht, the Netherlands
[2]Algorithms Group, TU Eindhoven, Eindhoven, Netherlands
[3]FernUniversität in Hagen, Hagen, Germany
[4]Edelsbrunner Group, IST Austria, Maria Gugging, Austria
[5]Freie Universität Berlin, Berlin, Germany

Submitted: November 2023 Accepted: October 2024 Published: November 2024
Article type: Regular paper Communicated by: M. A. Bekos and M. Chimani

Abstract. A face in a curve arrangement is called *popular* if it is bounded by the same curve multiple times. Motivated by the automatic generation of curved nonogram puzzles, we investigate possibilities to eliminate the popular faces in an arrangement by inserting a single additional curve. This turns out to be NP-hard; however, it becomes tractable when the number of popular faces is small: We present a randomized FPT-time algorithm where the parameter is the number of popular faces.

1 Introduction

Let $\mathcal{A}$ be a set of curves which lie inside the region bounded by a simple, closed curve, called the *frame*. All curves in $\mathcal{A}$ are either *closed*, or *open* with endpoints on the frame. We refer to $\mathcal{A}$ as a *curve arrangement*, see Figure 1a. We consider only *simple* arrangements, where no three curves meet in a point and there are only finitely many intersections in total, all of which are crossings (no tangencies).

Special issue on Selected papers from the Thirty-first International Symposium on Graph Drawing and Network Visualization, GD 2023

This work was initiated at the 16th European Research Week on Geometric Graphs in Strobl in 2019. A.W. has been supported by the Austrian Science Fund (FWF): W1230. S.T. has been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT19035] and by the NWO Gravitation project NETWORKS under grant no. 024.002.003. Part of the work was done while A.W. was emplyed at Graz University of Technology. Preliminary versions of this work have been presented at the 38th European Workshop on Computational Geometry (EuroCG 2022) in Perugia [10] and at the 31st International Symposium on Graph Drawing and Network Visualization (GD 2023) in Isola delle Femmine [11].

E-mail addresses: s.d.terziadis@tue.nl (Soeren Terziadis) alexandra.weinberger@fernuni-hagen.de (Alexandra Weinberger) zuzana.masarova@ist.ac.at (Zuzana Masárová) t.mtsentlintze@uu.nl (Tamara Mchedlidze) m.loffler@uu.nl (Maarten Löffler) rote@inf.fu-berlin.de (Günter Rote)

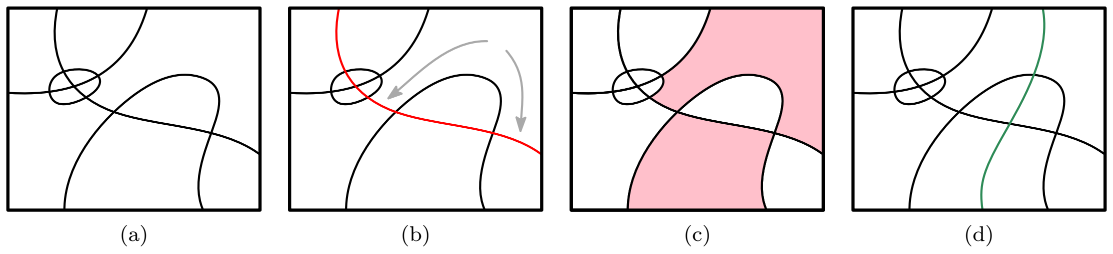


Figure 1: (a) A curve arrangement in a rectangular frame. (b) The top right face is incident to two disjoint edges of the red curve, making it *popular*. (c) All popular faces are highlighted. (d) After inserting an additional curve, the new arrangement does not contain any popular faces.

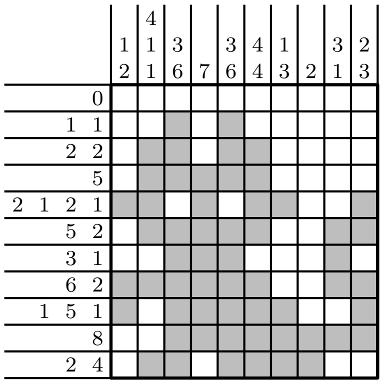
The arrangement $\mathcal{A}$ together with the frame can be seen as an embedded multigraph whose vertices are crossings of curves as well as the endpoints on the frame and whose edges are *curve segments* between these vertices. The arrangement $\mathcal{A}$ partitions the region bounded by the frame into *faces* (which are the faces of the embedded multigraph). A face is *popular* when it is incident to multiple edges belonging to the same curve in $\mathcal{A}$ (see Figures 1b–c). We study the following problem: can one additional simple curve ℓ be inserted into $\mathcal{A}$ such that no faces of $\mathcal{A} \cup \{\ell\}$ are popular (see Figure 1d)? We refer to this problem as Nonogram 1-Resolution (N1R) based on its motivation, which is explained in the next section.

In our arguments, we will often use the dual graph $\mathcal{A}^*$ of a curve arrangement $\mathcal{A}$, where every face of $\mathcal{A}$ is represented by a vertex and edge uv represents a shared curve segment between faces represented by u and v (not just a common crossing point). For ease of notation for a face f of $\mathcal{A}$ we label its corresponding vertex in $\mathcal{A}^*$ also with f. Note that $\mathcal{A}^*$ is an undirected multigraph without loops. A curve ℓ traversing $\mathcal{A}$ and crossing a sequence of faces $F_1, \ldots, F_k$ in that order can be expressed as a path $P = (F_1, \ldots, F_k)$ in $\mathcal{A}^*$. We will therefore also say that ℓ *contains* a vertex v or a path $v_1, \ldots, v_k$ in $\mathcal{A}^*$, implicitly using the expression of ℓ as a path in $\mathcal{A}^*$.

1.1 Nonograms

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 a column. A solved nonogram typically results in a picture (see Figure 2a). There is quite some work in the literature on the difficulty of solving nonograms [2, 4, 8].

Van de Kerkhof et al. [22] introduced *curved* nonograms, in which the puzzle is no longer played on a grid but on an arrangement of curves (see Figure 2b). In curved nonograms, clues specify numbers of filled faces of the arrangement in the sequence of faces incident to a common curve on one side. Van de Kerkhof et al. focus on heuristics to automatically generate such puzzles from a desired solution picture by extending curved segments on its boundary to form a curve arrangement.


(a)

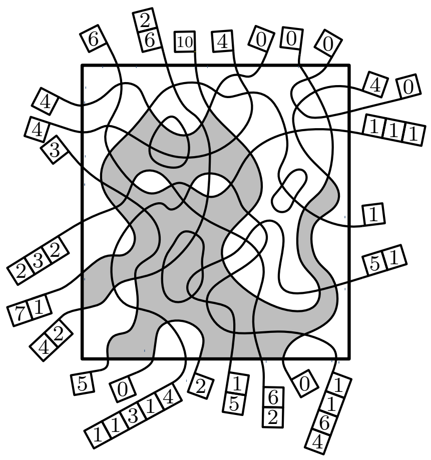

(b)

Figure 2: Two nonogram puzzles in solved state. (a) A classic nonogram. (b) A curved nonogram.

1.2 Nonogram complexity

Van de Kerkhof et al. observed that curved nonograms come in different levels of complexity — not in terms of how hard it is to *solve* a puzzle, but how hard it is to understand the rules (see Figure 4). They state that it would be of interest to generate puzzles of a specific complexity level; their generators can currently do this only by trial and error.

- *Basic* nonograms are puzzles in which each clue corresponds to a sequence of distinct faces. The analogy with clues in classic nonograms is straightforward.
- *Advanced* nonograms may have clues that correspond to a sequence of faces in which some faces may appear multiple times because the face is incident to the same curve (on the *same* side) 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. Expert nonograms are only suitable for experienced puzzle freaks [21].

Observation 1 *Arrangements with self-intersecting curves correspond exactly to expert puzzles.*

Proof: Every crossing point in a simple arrangement is incident to four faces (which are not necessarily distinct), see Figure 3. For each involved curve, let us arbitrarily label one of its two sides as the *left* side and the other one as the *right* side. Then, among the four incident faces, there will always be two faces for which the two incident sides carry the same label (*left*/*left* and *right*/*right*). The two other faces have mixed labels (*left*/*right* and *right*/*left*). In case of a self-intersection, the two mixed faces demonstrate that we have an expert puzzle.

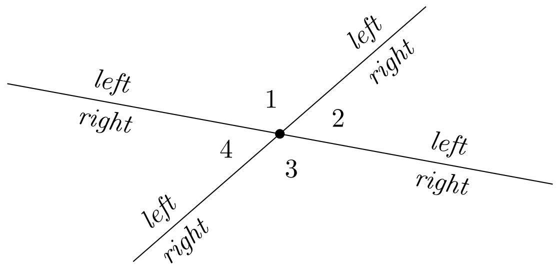


Figure 3: A crossing point in an arrangement with the four incident faces

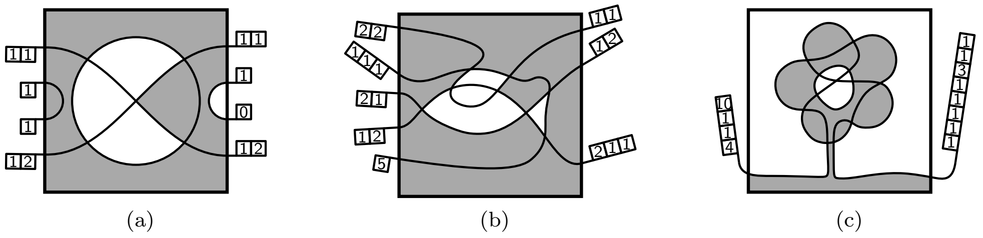


Figure 4: Three types of curved nonograms of increasing complexity [22], shown with solutions. (a) *Basic* puzzles have no popular faces. (b) *Advanced* puzzles may have popular faces, but no self-intersecting curves. (c) *Expert* puzzles have self-intersecting curves. We can see closed curves (without clues) in (a) and (c). All puzzles are part of the supplementary material of van de Kerkhof et al [22].

Conversely, a curve that does not cross itself cuts the frame into two parts, one on each side. This holds both for closed curves and for curves starting and ending on the frame. Hence, no face of the arrangement can be incident to both sides of such a curve. □

The difference between basic and advanced puzzles is more subtle; it is exactly the presence of *popular faces* in the arrangement.

One possibility to generate nonograms of a specific complexity would be to take an existing generator and modify the output. Recently, Brunck et al. [6] have investigated how popular faces in a nonogram might be removed by reconfiguring and/or reconnecting parts of curves at small neighborhoods, which they call switches (e.g. around curve crossings), and they have proved that it is NP-hard to decide if the curves can be rerouted and reconnected inside these local neighborhoods, such that the resulting arrangement has no popular faces. As an alternative, one may try to get rid of a popular face by adding extra curves that cut it into smaller pieces. If the resulting faces are not popular, we say that the popular face is *resolved*. Similarly we call the process of adding curves to a curve arrangement $\mathcal{A}$ until the resulting arrangement does not contain any popular faces as *resolving* $\mathcal{A}$.

Given Observation 1, it is obvious that an expert nonogram cannot be turned into an advanced or basic nonogram by adding additional curves: A self-intersection does not go away when additional curves are added. Therefore we focus on changing advanced puzzles into basic puzzles.

In this paper, we explore what we can do by inserting a *single* new curve into the arrangement. Specifically we focus on the following problem.

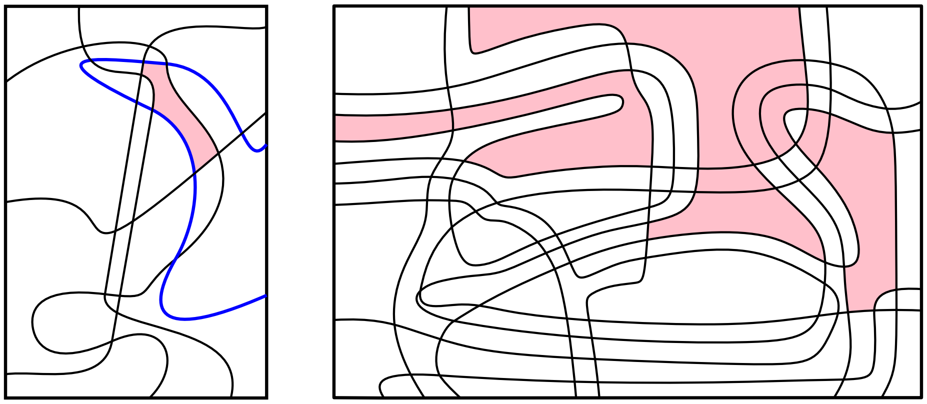


Figure 5: Puzzles (without clues) with all popular faces highlighted. The left image is taken from the supplementary material of van de Kerkhof et al [22]. The curve which appears twice on the boundary of a popular face is highlighted in blue in the left image.

Problem 1 (N1R) *Given a curve arrangement $\mathcal{A}$ with one or more popular faces in which all curves are not self-intersecting and start and end at a containing frame, decide if there exists a curve ℓ such that the curve arrangement $\mathcal{A} \cup \{\ell\}$ does not contain any popular faces.*

If not specified otherwise, we assume that ℓ also starts and ends at the frame.

1.3 Results

In Section 2, we discuss how the additional curve ℓ must pass through a *particular* popular face F in order to resolve it, in the sense that none of the faces into which F is subdivided by ℓ is popular. We show in Section 3 that deciding whether we can resolve all popular faces in a given curve arrangement by inserting a single curve – and therefore the N1R problem – is NP-complete. However, often the number of popular faces is small, see Figure 5. Hence, we are also interested in the problem parameterized by the number of popular faces k. We show in Section 4 that the problem can be solved (with a one-sided error) by a randomized algorithm in $O\left(2^k \text{poly}(n)\right)$ expected time. This algorithm involves calculations within a field of characteristic 2. The setup of a representation of this field involves a random element. Section 5 lays out how this can be done in a deterministic and more efficient manner. Our algorithm uses a modified dual graph, which represents popular faces as edge sets of size $O(n^2)$. Section 4.8 shows how this can be reduced to $O(n)$. Finally, we conclude with some open questions in Section 6.

2 Resolving a popular face by adding a single curve

As a preparation, we analyze how a single popular face F of an arrangement $\mathcal{A}$ can be resolved by adding a single new curve ℓ. If ℓ enters F and exits it again, we say that ℓ *visits* or *crosses* F. First, if ℓ visits F twice, this creates necessarily a popular face in $\mathcal{A} \cup \{\ell\}$. Therefore ℓ visits F only once and thus F is partitioned into two new faces in $\mathcal{A} \cup \{\ell\}$. Hence, if F contains more than two edges that belong to the same original curve, $\mathcal{A} \cup \{\ell\}$ necessarily contains a popular face (at least one of the new faces if ℓ visits F and F itself if not). Next assume F is popular because it has

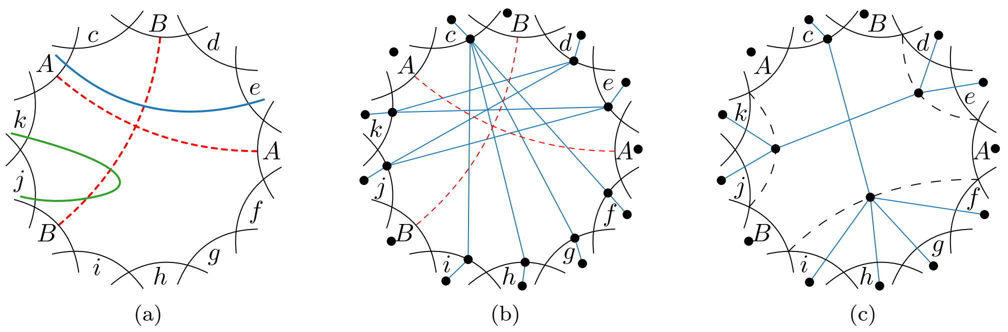


Figure 6: Resolving a popular face F. (a) Curtains model popular edges and two curves crossing F without resolving it properly. (b) All possible ways how ℓ can pass through F. (c) A more compact representation

two edges e, e' belonging to the same curve a. We call such edges *popular edges*. If ℓ crosses one of these edges, say e (like the blue curve in Figure 6a), then this partitions F into two faces, both of which contain an edge belonging to a (the edge that was part of e) and at least one of the faces contains another edge belonging to a (the edge that is e' or parts of it). Therefore $\mathcal{A} \cup \{\ell\}$ contains a popular face. And finally, since any curve of $\mathcal{A}$ can appear as an edge on the boundary of F at most twice, we can add a visual aid to F to indicate which edges are responsible for making F popular. Let e, e' be defined as above. We add a curve, which is entirely contained in F, starts at a point in the interior of e and ends at a point in the interior of e'. This curve is called a *curtain*. If ℓ crosses a curtain an even number of times or not at all (consider for an example the green curve in Figure 6a), then, since ℓ cannot enter and exit through neither e nor e', the start and the endpoint of the curtain are contained in the same new face created by the partition of F. Therefore $\mathcal{A} \cup \{\ell\}$ contains a popular face. Based on these observations we state the following lemma.

Lemma 1 *To ensure that a popular face F is resolved after the insertion of a single curve ℓ into the arrangement, it is necessary and sufficient that the curve ℓ has the following properties.*

a) *It visits the face F exactly once.*

b) *It does not enter or exit through a popular edge.*

c) *It separates each pair of popular edges, i. e., it crosses every curtain an odd number of times.*

In particular, suppose that $e_1, e_2, \ldots, e_k$ are consecutive edges of F, and e_1 and e_k belong to the same curve. Then a necessary condition for ℓ to resolve F is that

d) *ℓ crosses exactly one of the edges $e_2, \ldots, e_{k-1}$.* □

The ways how ℓ can traverse a popular face F can be modeled as a graph: We place a vertex on every edge of F except the popular edges. We then connect two such vertices u, v with an edge $e = uv$ if for every curtain c, the endpoints of c alternate with the vertices u and v around F, as shown in Figure 6b. This representation can be condensed as shown in Figure 6c and explained in Section 4.8.

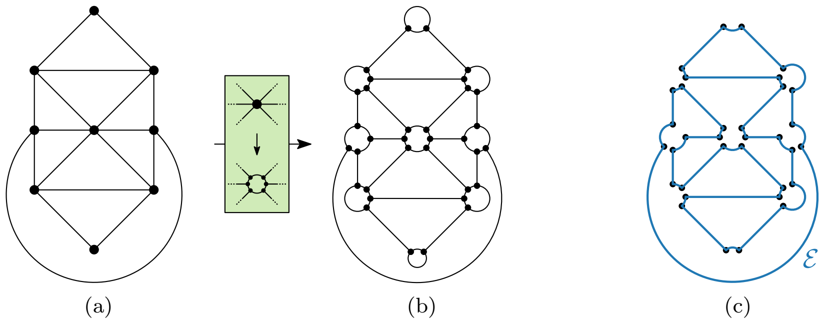


Figure 7: The vertices of the graph G (a) are replaced by cycles (b). A non-intersecting tour drawn on the modified graph visiting all original edges (c).

Observation 2 *Given a curve arrangement $\mathcal{A}$ in which all curves start and end at a containing frame, an additional curve ℓ that also starts and ends at the frame can be represented as a path in the dual graph $\mathcal{A}^*$ of $\mathcal{A}$. Conversely, all curves represented by a path in $\mathcal{A}^*$ traverse the same faces of $\mathcal{A}$ in the same order, even though their exact geometry can differ.*

3 N1R is NP-complete

In order to prove NP-hardness, we reduce from *Planar Non-Intersecting Euler Tour.* This reduction assumes ℓ to be a closed loop, but it can easily be adapted to work for an open curve ℓ' starting and ending at the frame.

3.1 Non-Intersecting Euler tours

An Euler tour in a graph is a closed walk that contains every edge exactly once.[1] An Euler tour in a graph embedded in the plane (a plane graph) is *non-intersecting* if every pair of consecutive edges $(a, b), (b, c)$ is adjacent in the radial order around b. Intuitively, an Euler tour is non-intersecting if it can be drawn without repeated vertices after replacing each vertex by a small cycle linking the incident edges in circular order (see Figures 7a and 7b). The Euler tour has to visit all of the original edges, but it does not have to cover the small vertex cycles (see Figure 7c).

The following problem was proved to be NP-complete by Bent and Manber [3, Theorem 1] by a reduction from Planar Satisfiability.

Problem 2 (Planar Non-Intersecting Euler Tour PNET) *Given a planar graph G embedded in the plane, decide whether G contains a non-intersecting Euler tour.*

A different NP-hardness reduction from the Hamiltonian cycle problem for planar 3-connected cubic graphs has been given by Fleischner [12, Section VI.3.3, pp. VI.156–9], see also [1, Theorem 2]. (In these references, non-intersecting Euler tours are studied extensively, referring to them by the term *A-trails* [12, Definition III.42a and Section VI.3].)

We alert the reader that there is a related notion, which is weaker: A *non-crossing* Euler tour is a tour that avoids crossings: A crossing at a vertex u is formed by two pairs of consecutive

[1] Alternative terms for Euler tours are Euler (or Eulerian) cycles or circuits or trails or lines.

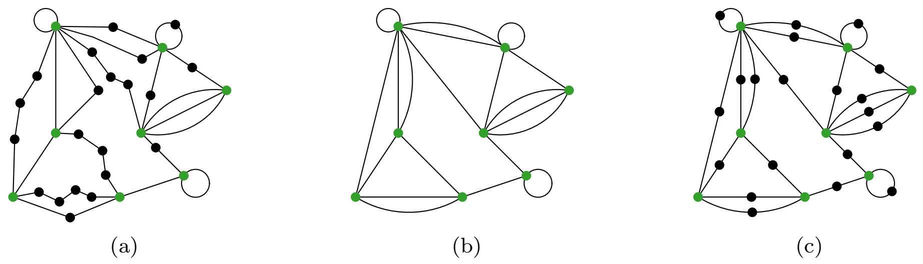


Figure 8: Preprocessing steps for the reduction. (a) The original input graph G of a PNET instance, which can be a multigraph with self-loops. Vertices of degree larger than 2 are highlighted in green. (b) All vertices of degree two are contracted. (c) All edges are subdivided once.

edges $(a,u),(u,b)$ and $(c,u),(u,d)$ in the tour such that their radial order around u is a,c,b,d or a,d,b,c. The distinction appears only for vertices of degree six or larger. Consider a vertex with six incident edges $e_1,\ldots,e_6$ in cyclic order. An Euler tour visiting these edges in the order $e_1,e_4,\ldots,e_3,e_2,\ldots e_5,e_6,\ldots$ does not have a crossing at u, but it is not non-intersecting.

It is an easy exercise to show that, in contrast to non-intersecting Euler tours, non-crossing Euler tours *always* exist, see [20]. A linear-time algorithm for constructing such an Euler tour has been recently described in [7, Lemma 4.2 of the full version], in the more general setting of graphs on arbitrary surfaces.

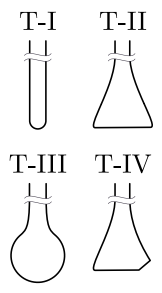
3.2 NP-completeness reduction

We will present a polynomial-time reduction from PNET to N1R, i. e., we will create a curve arrangement $\mathcal{A}$ containing popular faces based on a planar input graph G of PNET, such that there exists a curve ℓ for which $\mathcal{A}\cup\{\ell\}$ contains no popular faces if and only if G contains a non-intersecting Euler tour. We will refer to the curve ℓ as a *resolving curve*. We assume that G is connected and all vertices have even degree, because otherwise, G clearly cannot contain an Euler tour. We exclude the case that G is just a single cycle, because then PNET has a trivial solution. To make the arguments in the reduction easier, we modify G without affecting the existence of a planar non-intersecting Euler tour, see Figure 8. The modification is done in two steps. First, we replace every vertex of degree two with an edge connecting its neighbors, until no vertex of degree two is left; we then replace every edge with a path of length two. Every edge of G now connects a degree-2 vertex with a vertex of higher degree, and in particular, there are no self-loops. The drawing of G in the plane, which defines the order of the edges around each vertex, is inherited from the drawing of the original graph.

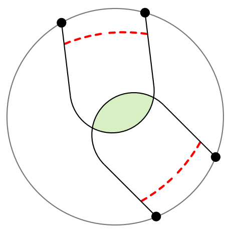
We place a rectangular frame around the drawing of G. We define the *outer face* as the intersection of the outer face of the drawing of G with the frame rectangle (which is now a bounded region). We represent every vertex $v\in V$ with a vertex gadget and every edge $e=uv\in E$ with an edge gadget.

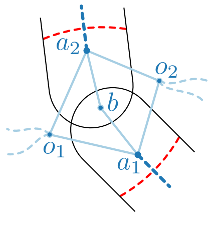
3.2.1 Vertex gadgets

The vertex gadgets consist of curves in one of four basic types of shapes, shown in Figure 9a, which we call beakers of Type I, Type II, Type III and Type IV, respectively. The exact geometric shape

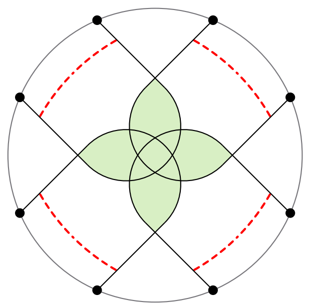


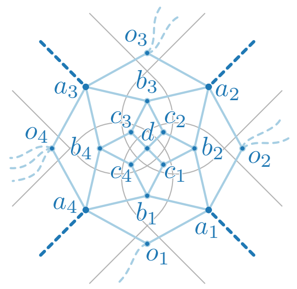
(a)


(b)

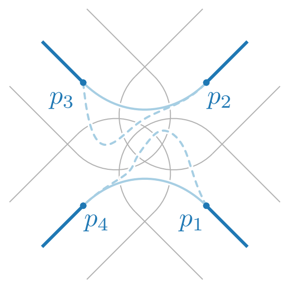


(c)


(d)



(e)



(f)

Figure 9: Construction of smaller vertex gadgets. The curtains in popular faces are shown as red dashed curves. (a) Basic beaker curve shapes. (b) Degree-two gadget (2 Type-I beakers) and (c) its dual graph. (d) Degree four gadget (4 Type-I beakers). (e) Dual graph of the degree 4 gadget. (f) A possible curve ℓ in light blue; some alternative routings, which connect the same endpoints, in dashed light blue. Faces highlighted in green are private to the gadget; the other faces are shared with other gadgets.

of these curves is not important, but having beakers of different shapes makes it easier to refer to them and their parts in our discussion. The important property is the combinatorial structure of the curve arrangements we create, as characterized by the dual graph.

We place the vertex gadget for a vertex v inside a circle around v, such that no two such circles intersect. We place one beaker to *represent* each incident edge e of v, at the position of v. The beakers are rotated such that their *bases* (the lower ends in Figure 9a) overlap in a specific pattern. The *opening* of each beaker (the upper ends in Figure 9a) points outwards, and it ends on the surrounding circle, to be continued outside into edge gadgets.

We use three variants of the vertex gadget, depending on the degree δ. In all cases, the endpoints of the δ beakers on the surrounding circle follow a regular pattern: The endpoints of a beaker are always adjacent in the circular order, see Figures 9b, 9d, and 10a below. Accordingly, as we go around the circle we have δ faces $a_1, \ldots, a_\delta$ that lie inside the beakers, separated by δ faces $o_1, \ldots, o_\delta$ between the beakers, see Figures 9c, 9e, and 10b. These 2δ faces are shared with other gadgets and with the free area outside the gadgets. All other faces are *private* faces of the gadget; they belong to no other gadget. We call the vertices a_i *beaker vertices*.

According to Observation 2, we will usually regard a resolving curve as a path in the dual graph.

Degree-two gadget. The vertex gadget for a degree-two vertex is simply made up of two overlapping Type-I beakers (see Figure 9b). The dual graph H_2 (shown in Figure 9c) of this construction is the complete bipartite graph $K_{2,3}$ with two beaker vertices a_1 and a_2, a private vertex b in the intersection of the beakers and two vertices o_1 and o_2 outside the beakers

Lemma 2 *Let $\mathcal{A}$ be a curve arrangement that contains a vertex gadget for a degree-two vertex and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ has to contain the path (a_1, b, a_2).*

Proof: By Lemma 1d, applied to the popular face a_1, ℓ has to contain the edge a_1b. A similar argument holds for the edge ba_2, and the statement follows. □

Degree-four gadget. The vertex gadget for a degree-four vertex v consists of four Type-I beakers, one per incident edge of v, which form the intersection pattern of Figure 9d.

It induces the following dual graph H_4, see Figure 9e. The center of the graph H_4 is a grid (three vertices by three vertices), labeled as follows. The center of this grid is labeled d, the vertices connected to the center are labeled in clockwise fashion c_i, $i \in \{1, 2, 3, 4\}$. The corners are labeled in a similar clockwise fashion as b_i, $i \in \{1, 2, 3, 4\}$. Consecutive corners of the grid b_i and b_{i+1} are connected to an additional vertex a_i, which is one of the beaker vertices. Finally the vertices a_{i-1} and a_i are connected to a vertex o_i. In this construction the beakers vertices are the four vertices a_1, a_2, a_3 and a_4.

Lemma 3 *Let $\mathcal{A}$ be a curve arrangement that contains a degree-four vertex gadget, and let ℓ be a resolving curve for $\mathcal{A}$. Then for every $i \in \{1, 2, 3, 4\}$, ℓ has to contain exactly one of the edges a_ib_i or a_ib_{i+1}. In particular, when it visits a_i, it must enter or exit the gadget.*

Proof: The first statement follows directly from Lemma 1d, applied to each popular face a_i. The second statement follows because ℓ cannot go from a_i to any of the other neighbors o_i and o_{i+1}, by Lemma 1b. □

The next lemma shows that the path of any possible resolving curve ℓ in the dual of the gadget contains two vertex-disjoint paths. The possible paths correspond to the ways how a non-intersecting Euler tour can connect edges incident to v in the original graph G.

Lemma 4 *Let $\mathcal{A}$ be a curve arrangement that contains a degree-four vertex gadget, and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ intersects H_4 in two vertex-disjoint paths P_1 and P_2 of the form*

a) $P_1 = (a_1, \dots, a_2)$ *and* $P_2 = (a_3, \dots, a_4)$ *or*

b) $P_1 = (a_1, \dots, a_4)$ *and* $P_2 = (a_2, \dots, a_3)$

Proof: By Lemma 3, the visit of ℓ to a_i must be an entry or exit to the gadget, and it must be paired with another exit or entry, which can only go through another beaker vertex a_j. If a_1 is paired with a_3, the remaining beaker vertices a_2 and a_4 must also be paired to each other, and this is impossible with two vertex-disjoint paths, by planarity. Only the two other matchings, as stated in the claim, remain. □

Lemma 4 states that a resolving curve has to contain two paths within H_4 connecting vertices in neighboring beakers. The actual path and sequence of faces crossed by the resolving curve is not (and is not required to be) unique. Different possibilities are shown in Figure 9f.

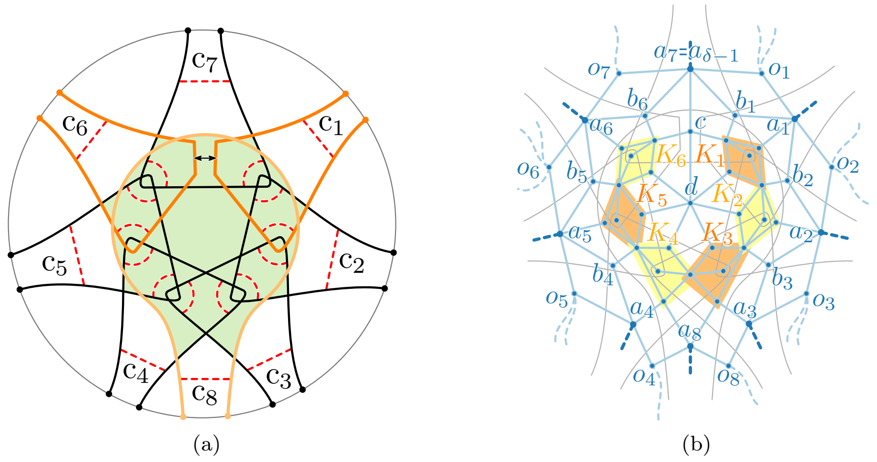


Figure 10: (a) Vertex gadget for a degree-8 vertex v with red dashed lines indicating the curtains of popular faces, and the surrounding circle. As before, the green faces are private to the gadget, and the other faces are shared. (b) Its dual graph. Dark blue dashed lines indicate further connections made by the edge gadget.

Degree-δ gadget. The vertex gadget for a vertex v of degree $\delta \geq 6$ is more complex and we first describe it informally, referring to Figure 10a. We place $\delta - 1$ Type-II beakers $c_1, \ldots, c_{\delta-1}$ symmetrically around the location of v. Each beaker intersects four adjacent beakers (two on each side), with the exception that the beakers $c_{\delta/2-1}$ and $c_{\delta/2+1}$ (dark orange curves in Figure 10a) do not intersect, which is indicated in Figure 10a with a small black two-headed arrow. Geometrically, we realize this by cutting a little piece from each of the beakers $c_{\delta/2-1}$ and $c_{\delta/2+1}$, turning them into Type-IV beakers.

We place an additional Type-III beaker c_δ (the light orange curve in Figure 10a) that surrounds all bases of the Type-II beakers and protrudes between $c_{\delta/2-1}$ and $c_{\delta/2}$, such that the intersection pattern of Figure 10a arises. More specifically, c_δ surrounds all intersection points between the other beakers except the outermost layer of intersection points of consecutive beakers c_i and c_{i+1} as well as between $c_{\delta-1}$ and c_1, but it does enclose both intersection points of $c_{\delta/2-1}$ and $c_{\delta/2}$.

We now define the gadget formally, by verbally describing the dual graph H_δ (see Figure 10b, and Figure 11a for an enlarged view of the central part). There is a central vertex d of degree $\delta - 1$. Around this central vertex there are $\delta - 2$ subgraphs isomorphic to $K_{2,3}$, which we call K_i for $i = 1, \ldots, \delta - 2$. They are embedded such that the vertices of the part of size 3 lie on a radial line through d. The two vertices of the other part are called p_i^l and p_i^r and lie to the left and right side of the radial line, respectively (when looking from d outwards). The innermost of the three vertices on the radial line is called q_i^d (because it is joined to d), the middle one q_i^c (for *central*), and the outer one q_i^a (because it is joined to some vertices a_j). For $i \in \{1, \ldots, \delta - 3\}$ the graph K_i is joined to K_{i+1} by identifying p_i^r and p_{i+1}^l, forming a chain of multiple $K_{2,3}$'s. The cycle is closed by an additional vertex c, which is connected to p_1^l and $p_{\delta-2}^r$. Additionally, c and the innermost vertex q_i^d of each part of size 3 of each K_i is connected to d. The vertices $p_1^l, \ldots, p_{\delta/2-1}^l$ are connected to a vertex $b_1, \ldots, b_{\delta/2-1}$, respectively. Similarly $p_{\delta/2}^r, \ldots, p_{\delta-2}^r$ are connected to $b_{\delta/2}, \ldots, b_{\delta-2}$. All these $\delta - 2$ vertices b_i lie outside the cycle of K_i. The beaker vertices are $a_1, \ldots, a_\delta$ (contained in the opening of the beaker). The vertices $o_1, \dots, o_\delta$ represent the space between beakers outside the gadget. The following additional edges appear in the gadget (and are also shown in Figures 10b and 11a).

- $a_i b_i$ for $i \in \{1, \dots, \delta - 2\}$
- $a_{i-1} b_i$ for $i \in \{2, \dots, \delta/2 - 1\}$
- $a_{i+1} b_i$ for $i \in \{\delta/2, \dots, \delta - 2\}$
- $a_{\delta-1} b_1$
- $a_i o_i$ for $i \in \{i, \dots, \delta\}$
- $a_{i-1} o_i$ for $i \in \{2, \dots, \delta - 1\} \setminus \{\delta/2\}$
- $a_\delta o_{\delta/2}$ and $a_{\delta/2-1} o_\delta$
- For $i \in \{1, \dots, \delta - 2\}$, a_i is connected to the outermost vertex q_i^a of the part of size three of K_i.
- $a_\delta p_{\delta-1}^a$ and $a_\delta p_\delta^a$

We now identify a set of edges that necessarily have to be contained in any resolving curve for a degree-δ vertex gadget. The goal of Lemmas 5–9 is to show that any resolving curve for an arrangement containing a degree-δ gadget has to traverse the gadget in one of two ways, which correspond to the two possibilities how a non-intersecting Euler Tour can connect the vertices incident to a degree-δ vertex.

Recall that $p_{i-1}^l = p_i^r$ and that all graphs K_i are made up of five vertices, i. e., p_i^l, p_i^r and a part of three vertices, one of which (q_i^d) is connected to d and one of which (q_i^a) is connected to one or multiple vertices a_j. The remaining vertex, q_i^c, is connected only to p_i^l and p_i^r.

Lemma 5 *Let $\mathcal{A}$ be a curve arrangement that contains a vertex gadget for a degree-δ vertex and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ has to contain the path $(p_1^l, q_1^c, p_2^l, q_2^c, \dots, p_{\delta-2}^l, q_{\delta-2}^c, p_{\delta-2}^r)$.*

Proof: Let f be the face represented by p_i^l. The edges $p_i^l q_i^a$ and $p_i^l q_i^d$ (highlighted orange in Figure 11a) cross the boundary f across segments belonging to the same curve, making f popular. By Lemma 1d, ℓ has to leave f between these two segments, and the only option to do so is by the edge $p_i^l q_i^c$. The same argument forces the inclusion of the edge $p_i^r q_i^c$. This holds for every K_i, resulting in the desired path (highlighted green in Figure 11a). □

For ease of reference, we refer to the endpoints of the path by $q = p_1^l$ and $q' = p_{\delta-2}^r$.

We will use the above lemma and Lemma 1 to reduce the edges in H_δ that can be traversed by ℓ. First, since by Lemma 1a, ℓ cannot enter a face twice, it can therefore also not use any vertex or edge in the dual twice. We can simplify H_δ by deleting the edges incident to any of the vertices $q_1^c, p_2^l, q_2^c, \dots, p_{\delta-2}^l, q_{\delta-2}^c$, i. e., vertices that are already incident to two edges in ℓ due to Lemma 5. Second, we can remove vertices of degree one in the remaining graph. And third, we can remove any edge corresponding to a resolving curve crossing a popular edge from the dual, since by Lemma 1b, a resolving curve cannot cross a popular edge. This include the edges $a_i o_j$ for any i and j in H_δ, as well as the edges cd, dq_1^d, $q_1^d p_1^l$, $dq_{\delta-2}^d$ and $q_{\delta-2}^d p_{\delta-2}^r$. Applying these three rules exhaustively yields the reduced dual graph H'_δ shown in Figure 11b.

The next lemma identifies sets of edges, of which exactly one needs to be contained in a resolving curve ℓ. It follows from a straightforward application of Lemma 1d to each of the faces a_i.

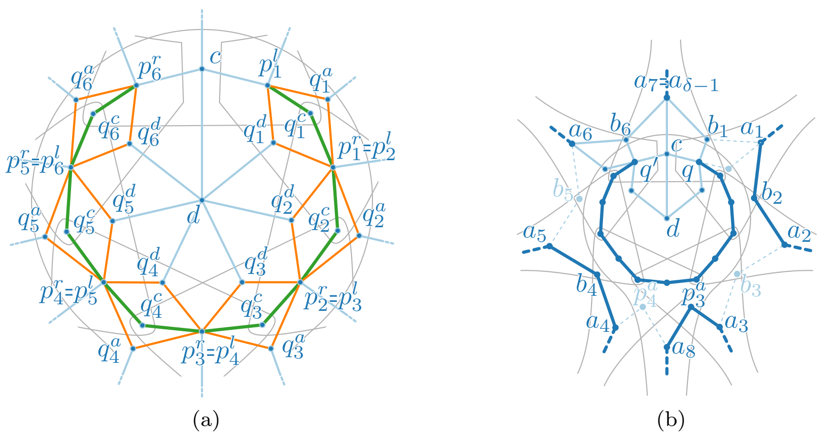


Figure 11: Detail of the center of the degree-δ vertex gadget. (a) Dual edges highlighted in orange cross boundary segments of faces that belong to curves that appear twice on the boundary of those faces. These edges cannot be used by a resolving curve. Green edges are forced by Lemma 5. (b) The reduced dual graph H'_8 with the forced part of ℓ due to Lemma 5 and one of the two options according to Lemma 7 shown in dark blue. Dark blue dashed lines indicate further connections made by the edge gadget.

Lemma 6 *Let $\mathcal{A}$ be a curve arrangement that contains a vertex gadget for a degree-δ vertex and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ has to contain exactly one edge of each of the following sets of edges:*

- a_1b_1, $a_1q_1^a$, *or* a_1b_2
- a_ib_i *or* a_ib_{i+1} *for* $i \in \{2, \ldots, \delta/2-2\}$
- $a_{\delta/2-1}b_{\delta/2-1}$ *or* $a_{\delta/2-1}q^a_{\delta/2-1}$
- $a_\delta q^a_{\delta/2-1}$ *or* $a_\delta q^a_{\delta/2}$
- $a_{\delta/2}b_{\delta/2}$ *or* $a_{\delta/2}q^a_{\delta/2}$
- a_ib_{i-1} *or* a_ib_i *for* $i \in \{\delta/2+1, \ldots, \delta-3\}$
- $a_{\delta-2}b_{\delta-2}$, $a_{\delta-2}q^a_{\delta-2}$, *or* $a_{\delta-2}b_{\delta-3}$
- $a_{\delta-1}b_1$, $a_{\delta-1}c$, *or* $a_{\delta-1}b_{\delta-2}$

Moreover, each visit of ℓ to one of the faces a_i is connected with an entry or exit from the gadget. □

To strengthen the previous lemma, we observe that the choices of these edges are not independent. In particular, we can notice the clockwise path $(b_2, a_2, b_3, \ldots, a_{\delta-2}, b_{\delta-2})$ (with some irregularity of notation in the middle), every odd member of which has degree 2.

Lemma 7 *Let $\mathcal{A}$ be a curve arrangement that contains a vertex gadget for a degree-δ vertex and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ has to contain exactly one the following two sets of paths.*

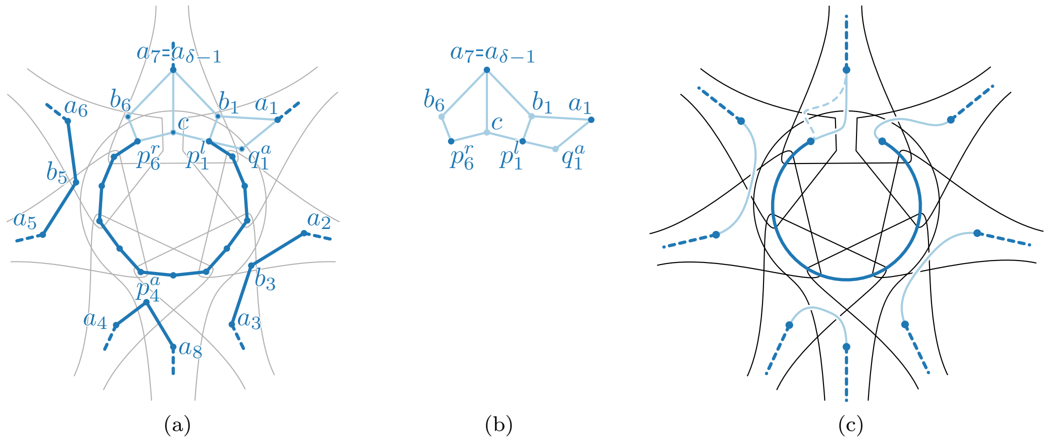


Figure 12: (a) One set of subpaths of ℓ forced by Lemma 7.1. (b) The subgraph of the dual of the vertex gadget that remains after removing unusable edges due to already forced subpaths of ℓ. (c) A resolving curve in the gadget. The dashed blue line indicates one alternative possibility for the sequence of traversed faces.

a) (a_i, b_{i+1}, a_{i+1}) *for even* $i \in \{2, \dots, \delta/2-2\}$, $(a_{\delta/2}, q^a_{\delta/2}, a_\delta)$ *and* (a_i, b_i, a_{i+1}) *for odd* $i \in \{\delta/2+1, \dots, \delta-3\}$.

b) (a_i, b_{i+1}, a_{i+1}) *for odd* $i \in \{1, \dots, \delta/2-3\}$, $(a_{\delta/2-1}, q^a_{\delta/2-1}, a_\delta)$ *and* (a_i, b_i, a_{i+1}) *for even* $i \in \{\delta/2, \dots, \delta-4\}$.

Proof: Observe that the vertices b_i for all $i \in \{1, \dots, \delta-2\}$ as well as $q^a_{\delta/2-1}$ and $q^a_{\delta/2}$ have degree 2. Therefore if ℓ contains one edge incident to one of these vertices it has to contain the second edge too, since, as stated above, no edge of $\mathcal{A}^*$ can be used twice. Since Lemma 6 forces either $a_\delta q^a_{\delta/2-1}$ or $a_\delta q^a_{\delta/2}$ to be included in ℓ, assume first that $a_\delta q^a_{\delta/2-1} \in \ell$. As a direct consequence $q^a_{\delta/2-1} a_{\delta/2-1} \in \ell$. Then neither $a_{\delta/2-1} b_{\delta-1}$ nor $b_{\delta-1} a_{\delta/2-2}$ can be included in ℓ. This propagates and forces all (a_i, b_{i+1}, a_{i+1}) to be included in ℓ for odd $i \in \{1, \dots, \delta/2-3\}$. The inclusion of $a_\delta q^a_{\delta/2-1}$ also makes the inclusion of $q^a_{\delta/2} a_\delta$ and therefore $a_{\delta/2} q^a_{\delta/2-1}$ impossible. This propagates and forces all (a_i, b_i, a_{i+1}) to be included in ℓ for even $i \in \{\delta/2, \dots, \delta-4\}$. With this all edges of the first case are included in ℓ. Assuming instead the inclusion of $a_\delta q^a_{\delta/2}$ in ℓ symmetrically proves the second case. $\square$

After considering the edges forced to be included in ℓ by Lemma 7, we are left with two symmetric possible states, one of which is shown in Figures 12a. The remaining part of the reduced dual graph is shown in Figure 12b. We know that ℓ must make an entry/exit through $a_{\delta-1}$ and another entry/exit through a_1, and since all other beaker vertices are taken, ℓ must connect $a_{\delta-1}$ to a_1 within the gadget. The other situation is symmetric, and we are therefore led to the following statement:

Lemma 8 *Let $\mathcal{A}$ be a curve arrangement that contains a vertex gadget for a degree-δ vertex and let ℓ be a resolving curve for $\mathcal{A}$.*

Then ℓ contains either the subpaths forced by Lemma 7.1 together with a path of the form $(a_1, \dots, a_{\delta-1})$, or it contains the subpaths forced by Lemma 7.2 together with a path of the form $(a_{\delta-2}, \dots, a_{\delta-1})$.

In both cases, there is such a system of paths such that all private faces are resolved.

Proof: In the first case, it is easy to route a path from a_1 to $a_{\delta-1}$, incorporating the subpath P between p^l_1 and $p^r_{\delta-2}$ that is forced by Lemma 5. One possibility for such a path is, e.g., concatenating either (a_1, b_1, p^l_1) or (a_1, q^a_1, p^l_1) with P and then either $(p^r_{\delta-2}, c, a_{\delta-1})$ or $(p^r_{\delta-2}, b_{\delta-2}, a_{\delta-1})$. Such a path eliminates all popular faces inside the gadget. The second case is symmetric. □

In fact, by looking at the subgraph from which the edges that are not yet fixed can be chosen (see Figure 12b), which has the vertices $p^l_1, p^r_{\delta-2}, a_1$ and $a_{\delta-1}$ on its outer face, one sees that the path must be of the form $(a_1, \dots, p^l_1, \dots, p^r_{\delta-2}, \dots, a_{\delta-1})$ or $(a_{\delta-2}, \dots, p^r_{\delta-2}, \dots, p^l_1, \dots, a_{\delta-1})$. There are four possible paths for either case.

We can now state the essential lemma for the variable gadgets. We say that ℓ *connects* two beakers of a gadget if there is a subpath of ℓ which is entirely contained within the dual of the gadget and does not contain any vertex representing another beaker. We also say that two beakers B_1 and B_2 are *neighbors* if, when traversing the boundary of the outer face in the dual of the gadget, the vertices representing B_1 and B_2 appear consecutively. The concept of neighboring beakers is also intuitively clear in the pictures of the gadgets in Figures 9b, 9d and 10a.

Lemma 9 *Let $\mathcal{A}$ be a curve arrangement that contains vertex gadgets and let ℓ be a resolving curve for $\mathcal{A}$. Then in any vertex gadget, ℓ connects vertices of neighboring beakers and there are exactly two such possibilities in every gadget.*

Proof: For the degree-two and the degree-four gadgets, this immediately follows from Lemma 2 and Lemma 4. For the gadget of vertices with degree six or larger note that all vertices representing beakers are covered by the paths forced by Lemma 7 and Lemma 8 and all forced paths connect neighboring beakers. □

Since there is one beaker placed in the gadget per edge incident to a vertex and the neighborhood relation between beakers mirrors the neighborhood relation of the edges, the two options of how subpaths of ℓ can traverse the dual of a vertex gadget directly correspond to the two options a planar non-intersecting Euler tour can traverse the original vertex. A possible resolving curve ℓ realizing one case of the forced traversal of the vertex gadget is shown in Figure 12c. Similarly to the degree-4 gadget, the exact set of faces traversed can slightly differ (indicated by the dashed line).

Recall that we have assumed at the start of this section that all vertex gadgets are contained in small circles – all of which are pairwise non-intersecting – around their respective vertex. We now remove these circles and are left with vertex gadgets whose curves all have two endpoints close to each other in the plane. In the following section we describe how we elongate these curves and, with the placement of some additional curves, form the edge gadgets.

3.2.2 Edge gadgets

Let $e = uv \in E$ be an edge in G and let B_u and B_v be the two beakers representing e in the vertex gadgets of u and v, respectively.

We now describe the construction of the edge gadget for the edge e. We will assume without loss of generality that e is drawn as a horizontal line. Similar to confining the vertex gadgets within a small disk around the vertex, we will construct the edge gadget within a horizontal strip.

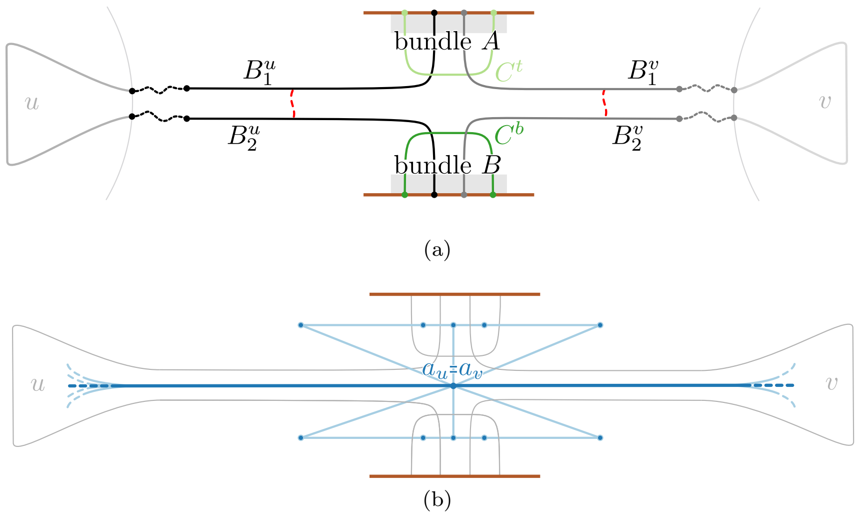


Figure 13: (a) Edge gadget for an edge $e = uv$. Open ends of all curves are collected into two bundles of parallel curves that lead into the incident faces beyond the brown line segments. Red dashed curtains indicate the popular face. (b) The dual graph of the gadget. The central vertex of the dual of the edge gadget is identified with one vertex from each vertex gadget. Edges connecting the central vertex to neighbors in the variable gadget are indicated as dashed light blue lines. An example of a subpath of ℓ is shown in dark blue.

The edge gadget consists of six curves, shown in Figure 13a; two each for the vertices u and v, which we call B_1^u, B_2^u, B_1^v and B_2^v, and two additional curves, which are called C^t and C^b. B_1^u and B_2^u (and similarly, B_1^v and B_2^v) are connected to the endpoints of the beakers representing e in the vertex gadgets of u and v, respectively, and belong therefore to the same curve. The curves B_1^u and B_1^v (B_2^u and B_2^v) are pairwise non-intersecting, start at a beaker and end on a horizontal line segment above (below) the gadget. The additional curve C^t starts on the horizontal line above the gadget left of the endpoint B_1^u and ends on the same line right of the endpoint of B_1^v. It intersects these two curves exactly once. As a result, the top boundary of the rectangle contains the endpoints of the following curves in order: C^t, B_1^u, B_1^v and the other endpoint of C^t. The second new curve C^b is placed symmetrically on the bottom. We call the collection of these curves a *bundle*. At this point of the construction, all used curves are either beakers in vertex gadgets or additional curves in an edge gadget. Therefore every open endpoint of every single curve is part of exactly one bundle.

The dual graph of this construction is shown in Figure 13b. It consists of two paths of length five and an additional central vertex, which is connected to the first, third and fifth vertex of either path. The only popular face in the edge gadget is the one represented by the central vertex. The central vertex is the same as the vertex a_u in the beaker representing e in the vertex gadget of u and the vertex a_v in the beaker representing e in the vertex gadget of v. These vertices represent a single face.

Lemma 10 *Let $\mathcal{A}$ be a curve arrangement that contains an edge gadget for an edge $e = uv$ and let ℓ be a resolving curve for $\mathcal{A}$. Then ℓ contains two edges connected to the central vertex of the edge gadget, whose other endpoint lies inside the vertex gadget of u and v, respectively.*

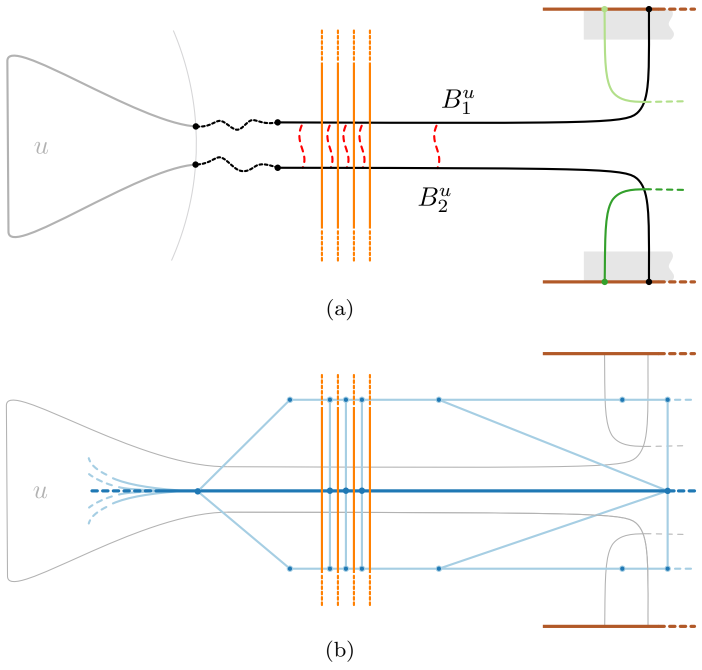


Figure 14: (a) A bundle (four orange vertical lines) originating at another edge gadget crosses the curves B_1^u and B_2^u as well as the popular face inside the edge gadget of an edge e. The face is partitioned into five new popular faces. (b) As a result, the dual graph contains a path between the beaker vertex of the beaker representing e in the vertex gadget and the central vertex of the edge gadget, and this path must traversed by a resolving curve.

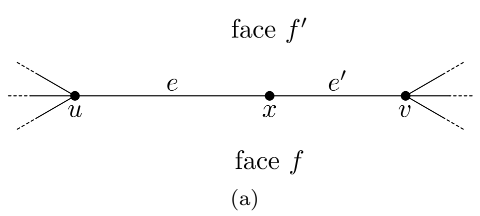
Proof: The statement follows immediately from the fact that in either of the two possibilities forced by Lemma 9 of how ℓ can traverse a vertex gadget, every beaker vertex a of a beaker representing an edge is an endpoint of exactly one subpath in ℓ. In particular this means that the central vertex of the edge gadget, which was unified with one such vertex of either vertex gadget is now incident to two edges. Since ℓ can visit the face represented by the central vertex at most once, no additional edge incident to the central vertex can be part of ℓ. □

Recall that so far we have placed a vertex gadget at the location of every vertex of G and an edge gadget along every edge of G and that every bundle starts in the middle of an edge gadget. Therefore the starting point of a bundle is in a face of the embedded graph G. It remains to connect the bundles (and therefore all open endpoints of all involved curves) to the frame.

3.2.3 Routing the bundles

The high-level idea is to route a bundle from face to face across other edge gadgets until it reaches the outer face, where all curves of the bundle are connected to the frame. First it is useful to observe that no curve appears twice on the outside of a bundle.

Observation 3 *The left- and right-most curve of every bundle is unique to that bundle, i. e., this curve does not appear in any other bundle.*



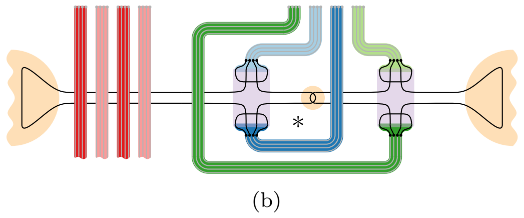


Figure 15: (a) In the graph G, two vertices u and v of high degree are connected via a path of length two containing two edges and the vertex x. (b) The curve arrangement contains two edge gadgets for the two edges e and e'. The bundles originating in the lower face f (shown in dark blue and dark green) can cross into f' without crossing their own edge gadget or each other. Other bundles (shown in red) can cross either edge gadget to get from f to f'. Vertex gadgets are highlighted in orange, edge gadgets in purple and the face marked with a star is the only face bounded by only three curves. The routing of the bundles is consistent with the directed edge ff' being contained in the shortest-path tree.

By contrast, a curve forming a beaker is part of exactly two bundles, both originating at the same edge gadget and leaving it in opposite directions. While it will not form a popular face inside the bundle, it would create a popular face if the two sides of the edge belong to the same face. The next observation show that this is impossible.

Observation 4 *Consider an edge e of G and the corresponding edge gadget. Then the two sides of the gadget belong to different faces of G.*

Proof: This follows from the fact that G, being an Eulerian graph, does not contain any vertices of odd degree. Therefore there is no edge that is incident to the same face on both sides. □

We will show below (Lemma 11) that a bundle can cross any edge gadget once without creating popular faces, *except that a bundle cannot cross the edge gadget from which it originated.*

We will now describe the sequence of faces through the bundles are led to the frame. We construct a shortest-path tree of the dual graph of G, rooted at the outer face. (Any other spanning tree will also do the job.)

For each bundle originating in a face f_1, the spanning tree contains a unique sequence $\mathcal{F} = f_1, \ldots, f_m$ of distinct faces, such that f_i and f_{i+1} share an edge and f_m is the outer face. We route the bundle through this sequence of faces. Since the outer face is incident to the frame, we can then simply connect the endpoints of the bundle to the frame.

Two bundles from different starting faces may meet in some face. Since the path in the tree continues in the same way for the two bundles, we can route them side by side in parallel, without having to cross them.

We still have to think of the exception stated above, i.e., that a bundle cannot cross the edge gadget of the edge where it originated. Here it is useful that all edges come in pairs e, e' connected by a degree-2 vertex. Suppose that a bundle originating in the gadget of e wants to cross this gadget from one incident face f to the other incident face f', because the shortest-path tree contains the edge ff'. (This can only happen immediately after starting the bundle.) In this case we can let the bundle cross from f to f' over the other edge gadget e', instead of e, as shown in Figure 15b. There will be another bundle that wants to cross from f to f', starting from e', and we route this bundle over e. In this way, none of the bundles needs to cross its own edge gadget or the other bundle.

As the final missing step, we establish a way for a bundle to cross an edge gadget without creating any popular faces.

Lemma 11 *Let $\mathcal{B}$ be a bundle originating at an edge e. Then $\mathcal{B}$ can cross the edge gadget of an edge e', $e' \neq e$ once without creating any popular faces outside of the edge gadget.*

Proof: Since $\mathcal{B}$ does not originate at the edge gadget of e', no curve of $\mathcal{B}$ is part of the edge gadget of e'. Therefore all edges of $\mathcal{B}$ can be drawn orthogonally across the two curves B_1^u and B_2^u as shown in Figure 14a (the case for B_1^v and B_2^v is symmetric). In doing so they cross the popular face represented by the central vertex of the edge gadget. The resulting dual graph, which accounts for one such bundle crossing, is shown in Figure 14b. Let a be the vertex in the dual of the vertex gadget that represents the beaker and c be the central vertex of the edge gadget. Instead of c and a being identified, a and c are connected with a path consisting of four edges and three degree four vertices. In general this path has four times as many edges as the number of bundles crossing the edge gadget next to each other, all internal vertices of these paths are of degree four and two of the four outgoing edges are popular. Since all of these vertices represent popular faces and in particular two of the outgoing edges of every internal vertex are popular, by Lemma 2 the entire path between a and c has to be contained in ℓ. Therefore a is still necessarily connected to the vertex a' representing the beaker on the opposite site of the edge gadget and the relevant property of the edge gadget is preserved. Further this path resolves all popular faces and no additional popular faces are created. □

This concludes the construction of our curve arrangement. We now observe the final necessary property of this arrangement stated in the following lemma.

Lemma 12 *All popular faces in $\mathcal{A}$ are contained in vertex gadgets and edge gadgets.*

Proof: All faces outside of these gadgets are either contained in a bundle, which by construction does not contain any popular faces or they are a part of a face of the drawing of G, which is partitioned by the bundles crossings this face. As a result, these faces are bounded by curves belonging to an edge gadget or the outside curves of bundles. Since by Observation 3, the outside curves of bundles are unique to the bundle, by Observation 4 the two sides of a bundle are incident to different faces and by construction two bundles do not cross. A bundle crossing an edge gadget twice is also impossible by construction.

The only edge case which needs to be considered is the single face which borders on the edge gadgets of degree two (marked in Figure 15b with a star), which is bounded by two beakers and a single bundle and therefore is also not popular. Therefore no popular faces are present outside of any gadget. □

3.2.4 NP-completeness.

We are now prepared to prove the main statement of this reduction, in particular, a polynomial-time reduction from PNET to N1R, which creates a curve arrangement $\mathcal{A}$ containing popular faces based on a planar input graph G of PNET, such that there exists a curve ℓ for which $\mathcal{A} \cup \{\ell\}$ contains no popular faces if and only if G contains a non-intersecting Euler tour.

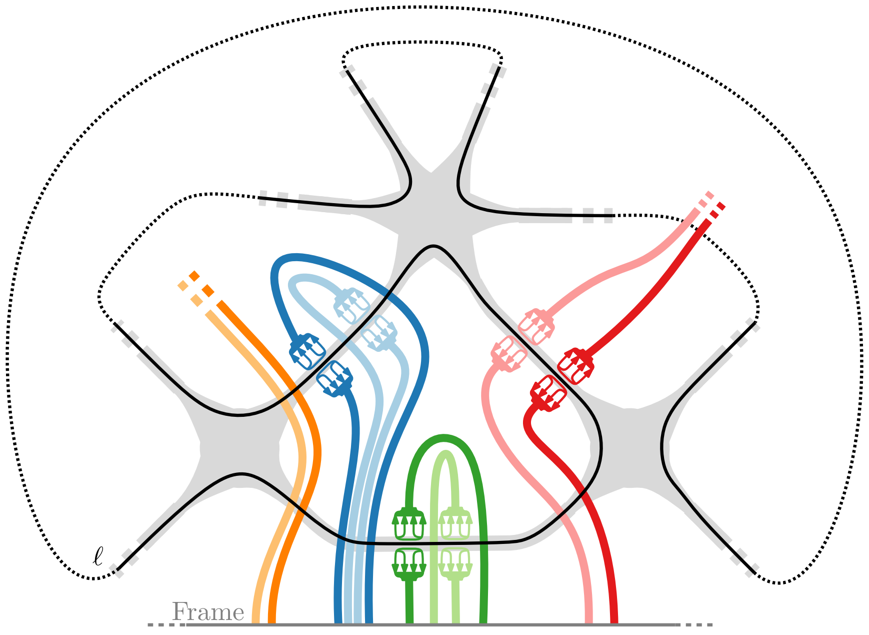


Figure 16: Schematic representation of three vertex gadgets and edge gadgets between them. The edges are subdivided by degree-2 vertices (not shown). The bundles are routed through beakers of other edges ending at the frame (partially shown at the bottom of the figure). Two additional bundles (shown in orange) are shown crossing over these edges. The red bundles to the upper right are routed on a different path through the construction. A possible routing of ℓ is shown as a black curve.

Theorem 1 *The variation of* N1R *where the goal is to find a resolving curve ℓ that is closed, i. e., it does not start and end at the frame, is* NP*-complete.*

Proof: Assume we are given a planar non-intersecting Euler tour $\mathcal{E}(G)$ of G as a cyclic permutation of the edges. We now show how to construct the curve ℓ. For every edge gadget follow a path according to Lemma 10. As a result ℓ consists at this point of a collection of paths, which resolve the popular faces in the edge gadgets and whose endpoints are beaker vertices in the vertex gadgets. Specifically every beaker vertex is an endpoint of such a subpath. Since $\mathcal{E}(G)$ connects only neighboring vertices in G, we connect the beaker vertices in the vertex gadgets in one of the two admissible patterns allowed by Lemma 9. Since every edge appears once in $\mathcal{E}(G)$ this results in one closed curve ℓ, which resolves all popular faces in the edge and vertex gadgets and by Lemma 12 no other popular faces are contained within the constructed arrangement.

Now assume we are given a resolving curve ℓ. By Lemma 10, ℓ has to traverse all edge gadgets. Additionally, to resolve all popular faces in the vertex gadgets, ℓ has to follow one of the two possible configurations of Lemma 9, which enforces that all consecutive edge gadgets traversed by ℓ correspond to neighboring edges of G. The order in which ℓ traverses all edge gadgets gives a permutation of all edges, which forms a planar non-intersecting Euler tour. We have shown that there exists a resolving curve ℓ if and only if G contains a non-intersecting Euler tour, and therefore N1R is NP-hard.

It is easy to see that N1R is in NP. The input arrangement can be represented as a plane graph in which the edges are marked as belonging to the different curves or to the frame boundary. The certificate is an extension of this arrangement by a resolving curve ℓ. Since ℓ cannot visit a face more than once, the certificate is of polynomial size. It can be easily verified in polynomial time whether the resolving curve ℓ is valid, in particular, whether the arrangement together with ℓ contains no popular faces. □

3.3 Adaptation to open curves

The reduction assumes that ℓ is a closed loop. It can be adapted to work for an open curve, i. e., we can create the arrangement $\mathcal{A}$, for which there exists a resolving curve ℓ' starting and ending at the frame if and only if G contains a planar non-intersecting Euler tour $\mathcal{E}(G)$.

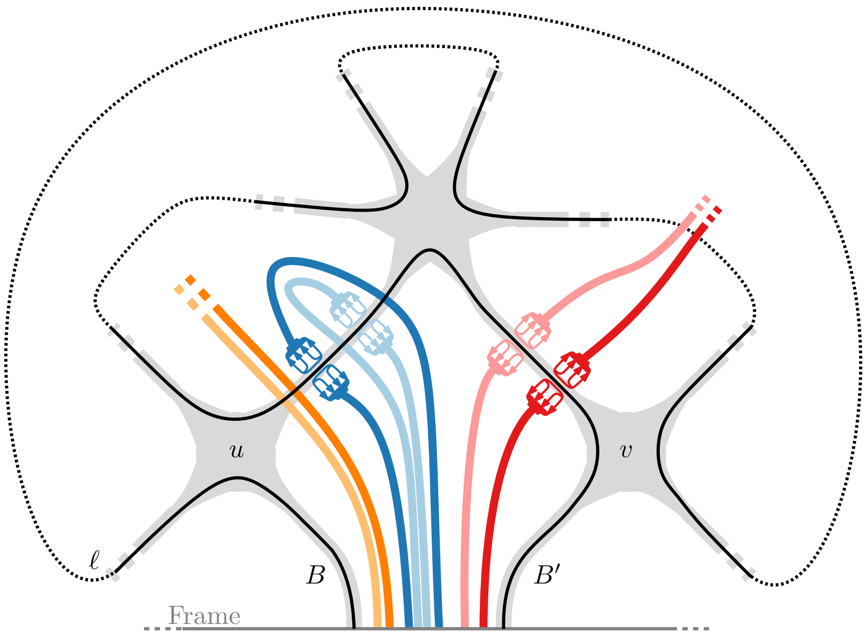


Figure 17: By routing both ends of an open beaker B (or B') in parallel to the frame instead of connecting them by an edge gadget to the extra vertex x, we force ℓ to reach the frame between the two connection points of the beaker.

Theorem 2 *N1R is* NP*-complete.*

Proof: The modified construction is illustrated in Figure 17. We select a path (u, x, v) between two high-degree vertices that is incident to the outer face. Instead of creating two edge gadgets for $e = ux$ and $e' = vx$, we eliminate the vertex x. The beaker B representing e in the vertex gadget of u and the the beaker B' representing e' in the vertex gadget of v are routed to the frame boundary.

Bundles that would otherwise have crossed the edge gadgets of e or e' to reach the frame can now connect directly to the frame between the two curves.

This does not create any additional faces (and in particular, no popular faces) except for the face in the opening of B and the face in the opening of B'. The boundary of the face in B contains three consecutive edges that are formed by B, the frame, and B again. Therefore, by Lemma 1d, ℓ must start on the frame between the two connection points of B, and similarly, it must end between the two connection points of B'. All other properties of the arrangement remain unchanged. $\square$

4 Randomized FPT-algorithm for N1R

In this section, we show that N1R with k popular faces can be solved by a randomized algorithm in $O\left(2^k \mathrm{poly}(n)\right)$ expected time, placing N1R in the class randomized FPT when parameterized by the number k of popular faces. We model N1R as a problem of finding a simple cycle (i. e., a cycle without repeated vertices) in a modified dual graph G, subject to a constraint that certain edges must be visited.

Problem 3 (Simple Cycle with Edge Set Constraints SCESC) *Given an undirected graph $G = (V, E)$ and k subsets $S_1, S_2, \ldots, S_k \subseteq E$ of edges, find a simple cycle, if it exists, that contains exactly one edge from each set S_i, for $i = 1, ..., k$. (The sets S_i do not have to be disjoint.)*

We mention that the SCESC instances that are used to solve the N1R problem will have disjoint sets S_i. In Problem 3 we do not exclude overlapping sets, as their treatment incurs hardly any overhead.

4.1 Correspondence between N1R and SCESC

We first show how an algorithm for Problem 3 with k edge sets may be used to solve N1R with k popular faces. We will use a modified dual graph of the curve arrangement as the input graph for SCESC.

To obtain the modified dual graph, we start with the dual graph of the given curve arrangement $\mathcal{A}$. Next we replace the vertex corresponding to the i-th popular face f with a set S_i of edges modeling the ways how an additional curve can cut all curtains of f, as described in Section 2 and shown in Figure 6b. To be specific, we place a *terminal vertex* on each edge s bounding f and connect it to the vertex of the face f' that is adjacent to f across s. (If f' is also popular, the vertex on e serves as a terminal vertex both in f and f'.) We connect two terminal vertices if a curve entering f through one edge and exiting through the other would cut all curtains of f. The latter connecting edges, which run through f, form the set S_i.

If we aim to obtain a closed resolving curve (analogous to Theorem 1), these adaptions are sufficient. To obtain a resolving curve, which necessarily starts and ends at the frame (analogous to Theorem 2), we add two additional vertices v_1^o and v_2^o in the interior of the outer face of the modified dual graph, and an edge e^o that connects v_1^o to v_2^o. Then we define the set $S_{k+1} = \{e^o\}$, i.e., we require that any resolving curve contains this edge. Finally we connect both vertices v_1^o and v_2^o to every vertex of a face f incident to the frame if f was not popular, or to the appropriate terminal vertex, which was placed on the frame edge if f was popular. This forces any closed resolving curve to cross the frame exactly twice. The single connected component of the resolving curve between these two crossing points with the frame is an open resolving curve starting and ending at the frame.

Lemma 13 *There is a correspondence between the simple cycles in the modified dual graph containing exactly one edge of every set S_i and the resolving curves for $\mathcal{A}$.*

Proof: As stated in Observation 2, a resolving curve corresponds to a simple cycle in the ordinary dual graph (before the introduction of the special edges S_i), because a resolving curve must pass from face to face without visiting a face more than once. A resolving curve necessarily visits each popular face in order to resolve that face, by Lemma 1. The possible ways how the curve can enter and exit the face in order to resolve it, according to Lemma 1, are encoded in the set S_i. Thus, if the cycle uses an edge of S_i, the corresponding popular face is resolved. The special Edge Set Constraints ensure that only one of the edges of S_i is used, i. e., the curve visits the popular face only once. □

We mention that the correspondence in Lemma 13 is even a one-to-one correspondence, apart from the fact that the geometric shape of the resolving curve can vary quite freely.

Based on Lemma 13, we can find a resolving curve by solving Problem 3. To this end, we will describe a randomized algorithm for this problem, extending an algorithm of Björklund, Husfeld, and Taslaman [5].

4.2 Solving SCESC

Theorem 3 a) *The SCESC problem on a graph with n vertices and $m \geq n$ edges can be solved in $O\left(2^k mn^2 \log \frac{2m}{n} \cdot \min_i |V(S_i)| \cdot W\right)$ expected time and $\Theta(2^k n + m)$ space with a randomized Monte-Carlo algorithm, with probability at least $1 - 1/n^W$, for any choice of the parameter $W \geq 1$. Here, $\min_i |V(S_i)|$ is the smallest size of the set of vertices used by any of the sets $S_1, \dots, S_k$. The model of computation is the Word-RAM with words of size $\Theta(k + \log n)$.*

b) *There is an alternative algorithm (Section 4.7) needing only polynomial space, namely $\Theta(kn + m)$, at the expense of an additional factor k in the runtime. It uses words of size $\Theta(\log n)$.*

Both algorithms find the cycle with the fewest edges if it exists (with high probability).

If $\mathcal{A}$ has n faces, the modified dual graph G has $O(n)$ vertices and $m = O(n^2)$ edges. The quadratic blow-up of m results from the construction as shown in Figure 6b. The number k of popular faces is the same as the number k of edge sets S_i.

With the alternative algorithm with polynomial space, since $k \leq n$, we get:

Corollary 1 *The N1R problem with k popular faces in a curve arrangement with n faces can be solved in expected time $O(2^k \text{poly}(n))$ and $O(kn)$ space.* □

The number of edges can even be reduced to $O(n)$. The details of this reduction process are discussed in Section 4.8 in more detail.

We first give a high-level overview of the algorithm. We start by assigning random weights to the edges from a sufficiently large finite field $\mathbb{F}_q$ of characteristic 2. Such a field exists for every size q that is a power of 2. In a field of characteristic 2, the law $x + x = 0$ holds, and therefore terms cancel when they occur an even number of times in a summation. The weight of a *walk* (with vertex and edge repetitions allowed) is the product of the edge weights of all visited edges. Our goal is now to compute the sum of weights of all closed walks, of given length, that satisfy the edge set constraints. The characteristic-2 property will ensure that the unwanted walks, those which are not simple, cancel, while a simple closed walk makes a nonzero contribution and leads to a nonzero sum with high property. The crucial idea is that, while these sets of closed walks can be very complicated, we can compute the aggregated sum of their weights in polynomial time. We have to anchor these walks at some starting vertex b, and we choose b to be one of the vertices incident to an edge of S_1.

More precisely, for each such vertex b, and for increasing lengths $l = 1, 2, \dots, n$, the algorithm computes the quantity $\hat{T}_b(l)$, which is the sum of the weights of all closed walks that

- start and end at b,
- have their first edge in S_1,
- use exactly one edge from each set S_i (and use it only once),
- and consist of l edges.

For the purpose of the following lemma, we consider the edge weights as variables and regard $\hat{T}_b(l)$ as a function of these variables. The result is a polynomial where each term is a product of l variables (possibly with repetition), and hence the polynomial has degree l, unless all terms cancel and it is the zero polynomial. We apply the following lemma, which is a straightforward adaptation of a lemma of Björklund et al. [5].

Lemma 14 a) *Suppose there exists a simple cycle of length l that satisfies the edge set constraints and that goes through an edge of S_1 incident to b. Then the polynomial $\hat{T}_b(l)$ is homogeneous of degree l and is not identically zero.*

b) *If there is no such cycle of length $\leq l$, the polynomial $\hat{T}_b(l)$ is identically zero.*

Proof: a) By assumption, there is a simple cycle among the walks whose weights are collected in $\hat{T}_b(l)$, and we easily see that the monomial corresponding to such a walk occurs with coefficient 1. Hence, $\hat{T}_b(l)$ is not identically zero. By definition, $\hat{T}_b(l)$ is a sum of weights of walks of length l, and hence it is clear that it is homogeneous of degree l.

b) For the second statement of the lemma, we have to show that $\hat{T}_b(l)$ is zero if there is no simple walk of length l or shorter. Since the field has characteristic 2, it suffices to establish a matching among those closed walks that satisfy the edge set constraints but do not represent simple cycles. Let W be such a walk. We will map W to another walk $\phi(W)$ that uses the same multiset of edges, by reversing (flipping) the order of the edges of a subpath between two visits to the same vertex v. Our procedure closely follows Björklund et al. [5], but we correct an error in their description. An example of the procedure, where the start vertex is $b = 1$, is shown in Equation System 1.

$$\begin{aligned} W = W_0 &= 1234156514323454617865 71 = 1[23415651432]345461786571 \\ W_0' = W_1 &= 1\underline{2}345461786571 = 123[454]61786571 \\ W_1' = W_2 &= 123\underline{4}61786571 = 1234[61786]571 \\ W &= 1234156514323454[61786]571 \\ \phi(W) &= 1234156514323454[68716]571 \end{aligned} \tag{1}$$

We look for the first vertex v that occurs several times on the walk. (During this whole procedure, the occurrence of b at the start of the walk is never considered.) We look at the piece

$[v \ldots v]$ between the first and last occurrence of v and flip it. If the sequence of vertices in this piece is not a palindrome, we are done. Otherwise, we cut out this piece from the walk and replace it with $[v]$. The resulting walk will still visit an edge from each S_i because such an edge cannot be part of a palindrome, since it is visited only once. Since the resulting walk W' is shorter than l, by assumption, it cannot be a simple cycle, and it must contain repeated vertices.

We proceed with W' instead of W. Eventually we must find a piece $[v \ldots v]$ that is not a palindrome. We flip it in the original walk W, and the result is the walk $\phi(W)$ to which W is matched. (The flipped piece $[v \ldots v]$ does not necessarily start at the first occurrence of v in the original sequence W, because such an occurrence might have been eliminated as part of a palindrome.[2])

It is important to note that after cutting out a palindrome $[v \ldots v]$, all repeated vertices must come *after* the vertex v. Hence, when the procedure is applied to $\phi(W)$, it will perform exactly the same sequence of operations until the last step, where it will flip $\phi(W)$ back to W.

Since the first edge of the walk is unchanged, the condition that this edge must belong to S_1 is left intact. This concludes the proof of Lemma 14. □

In case a) of Lemma 14, it follows from the Schwartz-Zippel Lemma [18, Corollary 1] (see also [17, Corollary Q1]) that, for uniformly random weights in $\mathbb{F}_q$, $\hat{T}_b(l)$ is nonzero with probability at least $1 - \text{degree}/|\mathbb{F}_q| = 1 - l/q \geq 1 - n/q$. We record this statement for later reference:

Proposition 15 *If the algorithm uses uniform random edge weights from the field $\mathbb{F}_q$, the failure probability is bounded by n/q.* □

Thus, if we choose $q > n^2$, we have a success probability of at least $1 - 1/n$ for finding the shortest cycle when we evaluate the quantities $\hat{T}_b(l)$ for increasing l until they become nonzero. The success probability can be boosted by repeating the experiment with new random weights.

In the unlikely case of a failure, the algorithm may err by not finding a solution although a solution exists, or by finding a solution that is not shortest. The last possibility is not an issue for our original problem, where we just ask about the existence of a cycle, of arbitrary length.

In the following, we will discuss how we can compute the quantities $\hat{T}_b(l)$, and the runtime and space requirement for this calculation. We describe the method for actually recovering the cycle after we have found a nonzero value in Section 4.6.

4.3 Computing sums of path weights by dynamic programming

We cannot compute the desired sums $\hat{T}_b(l)$ directly in an efficient manner, but have to do this incrementally via a larger variety of quantities $T_b(R, l, v)$ that are defined as follows:

For $R \subseteq \{1, 2, \ldots, k\}$ with $1 \in R$, $v \in V$, and $l \geq 1$, we define $T_b(R, l, v)$ as the sum of the weights of all walks that

- start at b,
- have their first edge in S_1,
- end at v,
- consist of l edges,
- use exactly one edge from each set S_i with $i \in R$ (and use it only once),
- contain no edge from the sets S_i with $i \notin R$.

[2]Here our procedure differs from the method described in [5], where the flipped subsequence extends between the first and last occurrence of v in W. In this form, the mapping ϕ is not an involution. The proof in [5], however, applies to the method as described here. This inconsistency was confirmed by the authors (Nina Taslaman, private communication, February 2021).

The walks that we consider here differ from the walks in $\hat{T}_b(l)$ in two respects: They end at a specified vertex v, and the set R keeps track of the sets S_i that were already visited. The quantities $\hat{T}_b(l)$ that we are interested in arise as a special case when we have visited the full range $R = \{1, \ldots, k\}$ of sets S_i and arrive at $v = b$:

$$\hat{T}_b(l) = T_b(\{1, \ldots, k\}, l, b).$$

We compute the values $T_b(R, l, v)$ for increasing values $l = 1, \ldots, n$. The starting values for $l = 1$ are straightforward from the definition.

To compute $T_b(R, l, v)$ for $l \geq 2$, we collect all stored values of the form $T_b(R', l-1, u)$ where (u, v) is an edge of G and R' is derived from R by taking into account the sets S_i to which (u, v) belongs. We multiply these values with the edge weight w_{uv} and sum them up. If (u, v) is in some S_i but $i \notin R$, we do not use this edge. Formally, let $I(u, v) := \{\, i \mid (u, v) \in S_i \,\}$ be the index set of the sets S_i to which (u, v) belongs. Then

$$T_b(R, l, v) = \sum_{\substack{(u,v) \in E \\ I(u,v) \subseteq R}} w_{uv} \cdot T_b(R \setminus I(u, v), l-1, u) \tag{2}$$

4.4 Finite field calculations

Each evaluation of the recursion (2) involves additions and multiplications in the finite field $\mathbb{F}_q$, where $q = 2^s > n$ is a power of 2.

We will argue that it is justified to regard the time for these arithmetic operations as constant, both in theory and in practice. The error probability can be controlled by choosing the finite field sufficiently large, or by repeating the algorithm with new random weights.

For implementing the algorithm in practice, arithmetic in $\mathbb{F}_{2^s}$ is supported by a variety of powerful libraries, see for example [16]. The size that these libraries conveniently offer (e.g., $q = 2^{32}$) will be mostly sufficient such that the success probability $1 - n/q$ guaranteed by Proposition 15 is satisfactory.

For the theoretical analysis, we propose to choose a finite field of size $q \geq n^2$, leading to a failure property less than $1/n$, and to achieve further reductions of the failure property by repeating the algorithm sufficiently often.

The following proposition describes some elementary approach for setting up the finite field $\mathbb{F}_q$ of size $q \geq n^2$, considering that we can tolerate a runtime and space requirement that is linear or a small polynomial in n. Various methods for arithmetic in finite fields $\mathbb{F}_{2^s}$ are surveyed in Luo, Bowers, Oprea, and Xu [14] or Plank, Greenan, and Miller [15]. The natural way to represent elements of $\mathbb{F}_{2^s}$ is as a polynomial modulo some fixed irreducible polynomial $p(x)$ of degree s over $\mathbb{F}_2$. The coefficients of the polynomial form a bit string of s bits. Addition is simply an XOR of these bit strings.

Proposition 16 *Given a positive integer n, one can set up data for a representation of a finite field $\mathbb{F}_q$ of characteristic 2 with $q > n^2$ elements in randomized $O(n \log^2 n)$ time and $O(n)$ space.*

With this representation, an element of $\mathbb{F}_q$ can be represented in $O(1)$ words, and two elements can be added or multiplied in $O(1)$ time.

These bounds hold for the Word-RAM model with words of size $\Theta(\log n)$, where arithmetic, logic, and addressing operations on words with $\Theta(\log n)$ bits are considered as constant-time operations.

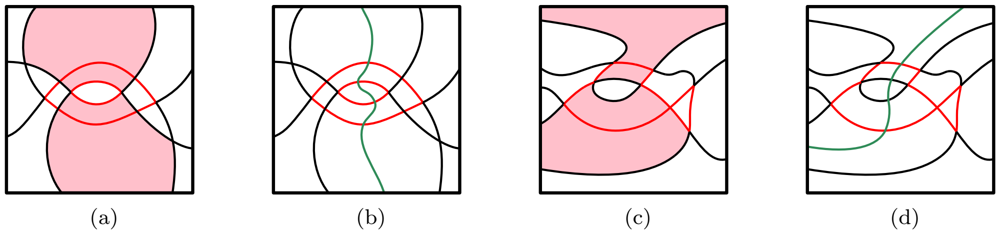


Figure 18: Input (a,c) and resulting output (b,d) generated by the implementation; the green curve resolves the popular faces with the smallest number of crossings.

Proof: We use the split table method, which is simple and implements multiplication by a few look-ups in small precomputed tables, see [14, Section 3.3.1] or [15, Section 6.5].

Let $C = \lceil(\log_2 n)/2\rceil$, and let $q = 2^{4C}$. Thus, $q \geq n^2$, as required.

According to [19, Theorem 20.2], an irreducible polynomial $p(x)$ of degree $d = 4C$ over $\mathbb{F}_2$ can be found in $O(d^4) = O(\log^4 n)$ expected time, by testing random polynomials of degree d for irreducibility. (Shoup [19] points out that this bound is not tight; there are also faster methods.)

Multiplication of two polynomials modulo $p(x)$ can be carried out in the straightforward "high-school" way in $O(d^2) = O(\log^2 n)$ steps: We form the product term by term and reduce the result by successive elimination of terms of degree larger than d.

We now consider the bitstring of length d as composed of $r = 4$ chunks of size C. In other words, we write the polynomial $q(x)$ as

$$q(x) = q_3(x)x^{3C} + q_2(x)x^{2C} + q_1(x)x^C + q_0(x),$$

where q_3, q_2, q_1, q_0 are polynomials of degree less than C. Addition takes $O(r)$ time, assuming an XOR on words of length $C = O(\log n)$ can be carried out in constant time. Multiplication is carried out chunk-wise, using $2r - 1 = 7$ multiplication tables. The j-th table contains the products $r(x)r'(x)x^{jC}$, for all pairs $r(x), r'(x)$ of polynomials of degree less than C, for $j = 0, 1, \ldots, 2r-2$. Multiplication in the straightforward way takes then $O(r^3)$ time. Each multiplication table has $2^C \times 2^C = O(n)$ entries of $r = 4$ words, and it can be precomputed in $O(nd^2) = O(n \log^2 n)$ time by multiplying in the straightforward way. □

The initial overhead of $O(n \log^2 n)$ is negligible in the context of our overall algorithm. We mention that, with another representation, the initial setup can even be made deterministic, and its time reduced to $O(n)$ (keeping the $O(n)$ space bound), see Section 5.

4.5 Runtime and space

The finite field additions and multiplications in (2) take constant time, as discussed in the previous section. Similarly, the set operation $R \setminus I(u,v)$ on subsets of $\{1, \ldots, k\}$ and the test $I(u,v) \subseteq R$ can be carried out in constant time, using bit vectors. Thus, for a fixed starting vertex $b \in V(S_1)$ and fixed R, going from $l-1$ to l by the recursion (2) takes $O(m)$ time in total, because each edge (u,v) appears in at most one of the sums on the right-hand side. The overall runtime is $O(|V(S_1)|2^k mn)$.

As mentioned after Proposition 15, the probability that the algorithm misses the shortest simple path is at most $1/n$. To amplify the probability of correctness, we repeat the computation t times, reducing the failure probability to $1/n^t$.

We consider each starting vertex b separately, and do not need to store entries for lengths $l-1$ or shorter when proceeding from l to $l+1$; thus the space requirement is $O(2^k n)$. For recovering the solution, the runtime must be multiplied by $O(n \log \frac{2m}{n})$, see Section 4.6. Figure 18 shows initial results of an implementation of our algorithm on two small test instances; see also [9].

4.6 Recovering the solution

The algorithm, as described so far, works as an oracle that only gives a yes-no answer (and a length l), but it does not produce the solution. To recover a solution, we will call the oracle repeatedly with different inputs.

Suppose the algorithm was successful in the sense that some number $\hat{T}_b(l)$ turned out to be nonzero, after finding only zero values for all smaller values of l. By Lemma 14, we conclude that there exists a simple cycle through b satisfying the edge set constraints, possibly (with small probability) shorter than l.

We will find a cycle by selectively deleting parts of the edges and recomputing $\hat{T}_b(1), \hat{T}_b(2), \ldots,$ $\hat{T}_b(l)$ for the reduced graph to see whether this graph still contains a solution. In this way, we will determine the successive edges of the cycle, and, as we shall see, we will know the cycle after at most $4n \log_2 \frac{4m}{n}$ iterations.

The first edge out of b is an edge of S_1, and thus we start by looking for the first edge among these edges. (The other edge incident to b, by which we eventually return to b, is not in S_1, and thus there is no confusion between the two edges incident to b.) In the general step, we have determined an initial part of the cycle up to some vertex u, and we locate the outgoing edge among the edges (u, v) incident to u, excluding the edge leading to u that we have already used.

In general, for a vertex u of degree d_u, we have a set of $d_u - 1$ potential edges. We locate a correct edge by binary search: We split the potential edges into two equal parts, and query whether a cycle still exists when one or the other part is removed. It may turn out (with small probability) that none of the two subproblems yields a positive answer. In this case, we repeat the oracle with new random weights. Since the success probability is greater than $1/2$, we are guaranteed to have a positive answer after at most two trials, in expectation. (We mention that such repeated trials may be necessary only when the length l for which we are looking is not the shortest length of a feasible cycle. Otherwise one can show, using arguments from the proof of Lemma 14, that the polynomial for the original problem is the sum of the polynomials for the two subproblems. Thus, at least one of the two subproblems must give a positive answer.) After at most $2\lceil \log_2(d_u - 1) \rceil$ successful queries, we have narrowed down the search to a single outgoing edge uv, and we continue at the next vertex v.

For a walk W of length l, we use, in expectation, less than $Q := \sum_{u \in W} 4(1 + \log_2 d_u)$ queries, where $\sum_{u \in W} d_u \leq 2m$. By Jensens inequality, Q can be bounded by

$$Q \leq 4l \left(1 + \log_2 \tfrac{2m}{l}\right) \leq 4n \left(1 + \log_2 \tfrac{2m}{n}\right) = 4n \log_2 \tfrac{4m}{n},$$

as claimed.

During this procedure, it may also turn out that a solution with fewer than l edges exists. In this case, we know that we must have been in the unlikely case that the original algorithm failed to produce a nonzero value for the shortest solution l. We simply adjust l to the smaller value and continue.

Note that this procedure is guaranteed to produce a simple cycle, although not necessarily the shortest one. The algorithm selects a branch only when the corresponding polynomial is nonzero, which implies that a simple cycle exists in that branch.

Some simplifications are possible. For edges that are known to belong to every solution (for example if some set S_i contains only one edge), we can assign unit weight, thus saving random bits, reducing the degree of the polynomial, and increasing the success probability. Edges that would close a loop can be discarded. When the graph is sparse and $m = O(n)$, we can simply try the $d_u - 1$ edges one at a time instead of performing binary search, at no cost in terms of the asymptotic runtime.

4.7 Reduction to polynomial space

As described, the algorithm has an exponential factor 2^k in the space requirement. This exponential space requirement can be eliminated at the expense of a moderate increase in the runtime, by using an inclusion-exclusion trick that was first used by Karp [13] for the Traveling Salesman Problem. The possibility of applying this trick in the context of our problem was already mentioned in Björklund et al. [5].

For a "forbidden set" $F \subseteq \{2, 3, \ldots, k\}$, $v \in V$, $1 \le l \le n$, and $1 \le j \le k$ let $U_b(F, l, j, v)$ be the sum of the weights of all walks that

- start at b,
- have their first edge in S_1,
- end at v,
- consist of l edges,
- contain no edge from the sets S_i with $i \in F$,
- use in total j edges from the sets S_i, counted with multiplicity: If an edge belonging to p different sets S_i is traversed r times, it contributes pr towards the count j.

The clue is that we can compute these quantities for each F separately in polynomial time and space, by simply removing the edges of S_i for all $i \in F$ from the graph. The dynamic-programming recursion is straightforward. In contrast to (2), we have to keep track of the number j of edges from $S_1 \cup \cdots \cup S_k$.

We regard each index $i \in \{2, \ldots, k\}$ as a "feature" that a walk might have (or an "event"), namely that it avoids the edges of S_i. By the inclusion-exclusion formula, we can compute the sum of weights of paths that have none of the features, i. e., that visit *all* sets S_i. In this way, we get the following formula:

Lemma 17
$$\hat{T}_b(l) = \sum_{F \subseteq \{2,3,\ldots,k\}} (-1)^{|F|} U_b(F, l, k, b) \tag{3}$$

Proof: The parameters l and b match on both sides. By the inclusion-exclusion theorem, the right-hand side is the sum of all walks in which every set S_i is visited *at least* once. Since the parameter j is equal to k, we know that there were only k visits to sets S_i. where membership of an edge in several sets S_i has been duly taken into account. Thus, every set S_i is visited *exactly* once. □

The sign $(-1)^{|F|}$ is of course irrelevant over a field of characteristic 2.

To reduce the space requirement as much as possible, we organize the computation as follows. First, each starting point b is considered separately. We initialize n variables for accumulating the contributions to the quantities $\hat{T}_b(l)$ according to (3). Then, for each forbidden set F separately, we compute the quantities $U_b(F, l, j, v)$ for all j and v, incrementally increasing $l = 1, 2, \ldots, n$. Since we need to remember only the entries for two consecutive values l at a time, this requires only $O(nk)$ space. Along the way, we add the contributions $U_b(F, l, k, b)$ to (3).

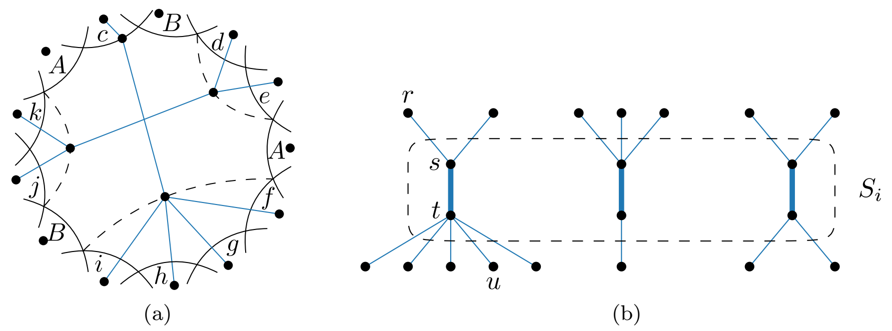


Figure 19: A more compact representation that avoids the quadratic blow-up. (a) The representation using central edges (repeated from Figure 6c). (b) A set S_i and its connecting edges

In summary, we can calculate $\hat{T}_b(1), \hat{T}_b(2), \ldots$ in space $O(nk)$. Compared to the exponential-space algorithm, we need an additional factor k in the runtime, for the k choices of the parameter j.

4.8 Adapting the Edge Set Constraints

In Section 2, we have mentioned a modified construction of the graph G on which a nonintersecting Euler tour is sought, which avoids the quadratic blow-up of the number of edges.

Here we give more details of this construction, see the example in Figure 19a. After removing the popular edges on the boundary of the face, the boundary edges are partitioned into *runs* of consecutive edges, like the run $R = (f, g, h, i)$ in the example. The original construction places a terminal node on every non-popular edge; the quadratic blow-up results from connecting all edges from one run with all edges from another run. We avoid this by choosing a representative node v_R for each run to replace the nodes on the individual boundary edges. (Runs of length one can stay as they are.) Geometrically, we can imagine inserting a chord in the face (shown as a dotted curve), overarching the edges of the run, on which the new terminal node v_R is placed.

The edges between all nodes of two runs of the original construction are replaced by a single edge between the respective representative nodes of the runs. We call these edges *central edges*. For each non-popular boundary edge, the graph has an edge that crosses it, connecting some representative node with the rest of the graph. We call these edges the *peripheral edges*. The size of this construction is clearly linear in the size of the face.

One has to take care that the cycle that is found does not use two peripheral edges in succession, like the edges crossing f and h, because such a cycle would not correspond to a valid curve. This constraint must be added to the problem definition, and the recursion (2) must be modified accordingly.

The sets S_i have a special structure. Each set S_i consists of the vertex-disjoint central edges of a popular face (the thick edges in Figure 19b). The peripheral edges, which connect the central edges to the remainder of the graph, do not belong to S_i. We do not want to consider cycles that use two peripheral edges (incident to the same S_i) in succession, without going through the central edge. We impose this as an extra condition on the walks whose weights we accumulate.

It is easy to incorporate this condition in the recursion of Section 4.3: Instead of the quantities $T_b(R, l, v)$, we work with quantities $\tilde{T}_b(R, l, v, p)$ that depend on an additional parameter p. This is a binary variable that tells whether the last edge of the walk has traversed a peripheral edge in the direction towards the central edge. If this is the case, we force the walk to use the central edge in the next step.

In this way, the additional condition incurs a blow-up of at most a factor 2 in the size of the dynamic programming tables and in the runtime.

We now argue that Lemma 14 still holds for these modified quantities. The proof of Lemma 14 goes through for the following reason. The conditions on allowed walks ensure that an endpoint s of a central edge (s, t) (see Figure 19b for an example) is always part of a subpath consisting of the central edge surrounded by two peripheral edges, (like (r, s, t, u) or (u, t, s, r) in Figure 19b). Such a subpath cannot be part of a palindrome, since (s, t) belongs to a special set S_i, and for the same reason, s can never be a repeated vertex of a walk. If the bijection constructed in the proof reverses a subpath that goes through the vertex s, this is not a problem because the reversed traversal does not violate the extra condition.

5 Variation: Deterministic setup of finite field computations in characteristic 2

In Section 4.4, we have described how to obtain a field of characteristic 2, how to represent it and how to perform calculations with this representation. While this is good enough for our application, it is not optimal when the task is viewed in isolation. In this section, we present an alternative procedure which is better on two accounts: (i) While the procedure of Section 4.4 is randomized, because it involves testing of random polynomials for irreducibility, the alternative method is deterministic. (ii) The entire setup requires an initial time investment of $O(n \log^2 n)$. We show that this can be reduced to $O(n)$, while keeping the space requirement of the auxiliary tables and data structures down to $O(n)$. While this result is beside the main point of our paper, we include it for reference, because it might be useful in other contexts.

Theorem 4 *Given a positive integer n, one can set up data for a representation of a finite field $\mathbb{F}_q$ of characteristic 2 with $q > n^2$ in deterministic $O(n)$ time and space in the Word-RAM with words of size $\Theta(\log n)$. With this representation, an element of $\mathbb{F}_q$ can be represented in $O(1)$ words, and two elements can be added or multiplied in $O(1)$ time.*

The remainder of this section is a detailed explanation of the construction necessary to prove Theorem 4. The construction proceeds in several stages, which are described in the following sections.

5.1 Step 1: Getting a primitive polynomial in quadratic time

There is a completely naive algorithm for constructing $\mathbb{F}_q$ in $O(q^2)$ time and $O(q)$ space, for $q = 2^s$, assuming q fits in a word (s bits). Simply try out all polynomials $p(x)$ over $\mathbb{F}_2$ of degree s. There are $2^s = q$ possibilities.

For each $p(x)$, try to construct the logarithm table ("index table") in $O(q)$ steps by trying whether the polynomial x generates the nonzero elements of $\mathbb{F}_2[x]$: Start with the string $00\dots01$ representing the polynomial 1, and multiply by x by shifting to the left, and adding $p(x)$ (XOR with the corresponding bit string) to clear the highest bit if necessary. Repeat $q - 2$ times and check whether $00\dots01$ reappears. If not, then we are done.

This will be successful if and only if $p(x)$ is a primitive polynomial modulo 2. Since primitive polynomials exist, we are guaranteed to find an index table in $O(q^2)$ time. With the help of the index table and its inverse (the antilogarithm table), multiplication in $\mathbb{F}_q \setminus \{0\}$ is reduced to addition modulo $q-1$ with the help of three table look-ups.

Actually, for testing whether $p(x)$ is irreducible, it is sufficient to check the powers x^k of x where k is a maximal proper divisor of $q-1$. The primes dividing $q-1$ can be trivially found in $O(\sqrt{q})$ time, and there are less than $\log q$ of them. Computing x^k takes $O(s^2 \log q) = O(\log^3 q)$ time if the $O(s^2)$ time schoolbook method of multiplication modulo $p(x)$ is applied, together with repeated squaring to get the k-th power. Thus, we need $O(q \log^3 q)$ time instead of $O(q^2)$. This is probably still a gross overestimate because primitive polynomials modulo 2 are frequent: There are at least $q/(2s)$ of them [19, Theorem 19.11].

5.2 Step 2: From $\mathbb{F}_q$ to $\mathbb{F}_{q^2}$ in $O(q)$ deterministic time and space

Assuming that we can do arithmetic in the field $\mathbb{F}_q$, we show how to do arithmetic in $\mathbb{F}_{q^2}$. This is achieved by a degree-2 field extension. We look for an irreducible polynomial of the form $p(x) = x^2 + x + p_0$ over $\mathbb{F}_q$. If this polynomial were reducible, we could write

$$p(x) = (x+a)(x+b) = x^2 + (a+b)x + ab.$$

Due to restriction of $p(x)$ to a special form, it follows that $a+b=1$, and hence $b = 1+a$, and

$$p(x) = (x+a)(x+(a+1)) = x^2 + x + a(a+1).$$

The expression $a(a+1)$ can take at most $q/2$ different values, because $a=c$ and $a=c+1$ lead to the same product, since $(c+1)+1=c$. Thus, we can group the q potential values a into $q/2$ pairs with the same product, and at least $q/2$ product values must be unused. We can find the range of $a(a+1)$ by marking its values in an array of size q. Any unmarked value can be used as the constant term p_0. This takes $O(q)$ time and space.

Multiplication of two polynomials $a_1x + a_0$ and $b_1x + b_0$ gives the following product.

$$\begin{aligned}
c_1x + c_0 &= (a_1x + a_0)(b_1x + b_0) \\
&= a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0 \\
&= a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0 - a_1b_1(x^2 + x + p_0) \\
&= (a_1b_0 + a_0b_1 + a_1b_1)x + (a_0b_0 + a_1b_1p_0)
\end{aligned}$$

Multiplication can therefore be carried out as follows:

$$\begin{aligned}
c_1 &= a_1b_0 + a_0b_1 + a_1b_1 \\
&= (a_1 + a_0)(b_1 + b_0) - a_0b_0 \\
c_0 &= a_0b_0 + a_1b_1p_0
\end{aligned}$$

with four multiplications in $\mathbb{F}_q$ (and four additions), using the common term a_0b_0 for both coefficients.[3]

[3]This is the same trick as for multiplying two complex numbers with three multiplications instead of four. By contrast, [15, Section 6.8] propose irreducible polynomials of the form $x^2 + p_1x + 1$, leading to five multiplications, one more than our method. However, one must be careful in this comparison: In cases where multiplications are done by index tables, the mere number of multiplication operations does not determine the runtime alone; a common factor that appears in several multiplications saves lookup time in the logarithm tables.

5.3 Step 3: Setup of a finite field $\mathbb{F}_q$ with $q > n^2$ in deterministic $O(n)$ time and space

Let $C = \lceil(\log_2 n)/2\rceil$. First we construct $\mathbb{F}_q$ for $q = 2^C = O(\sqrt{n})$ in $O(q^2) = O(n)$ time, as described in Section 5.1. To carry out multiplications in $\mathbb{F}_q$ in constant time, we can store a logarithm and antilogarithm table, in $O(q)$ space, or we can even compute a complete multiplication table, in $O(q^2) = O(n)$ time and space.

Then, by the method of Section 5.2, we go from $q = 2^C$ to $q = 2^{2C}$, and finally from $q = 2^{2C}$ to $q = 2^{4C}$, in $O(2^{2C}) = O(n)$ time and space. Then addition and multiplication in $\mathbb{F}_{2^{4C}}$ can be carried out in constant time. Multiplication goes down two recursive levels, from $q = 2^{4C}$ via $q = 2^{2C}$ to $q = 2^C$, before the 16 resulting multiplications are resolved by table look-up. This proves Theorem 4.

In order to speed up the process, we can eliminate the lower level of recursion by building a log/antilog table for $q = 2^{2C} = O(n)$ of size $O(n)$, to do the multiplication by table look-up already at this level. To prepare the tables, we need a generating element of $\mathbb{F}_{2^{2C}}$. Such a generating element can be constructed in $O(2^{2C}) = O(n)$ time, as shown in the next section 5.4.

5.4 Constructing an index table without knowing a generator

We assume that multiplication in $\mathbb{F}_q$ takes constant time, but no generating element for the multiplicative group of $\mathbb{F}_q$ is given. Our goal is to construct logarithm and antilogarithm tables (index tables), of size $q-1$, in time $O(q)$. In order to generate these tables, we need a generating element.

Essentially, we have to find a generator of a cyclic group, which is given by a multiplication oracle and the list of its elements.

We start with an arbitrary element $a_1 \neq 0$ and try to construct the index table by running through the powers of a_1. We mark the elements that we find. If the process returns to 1 before marking all nonzero elements, we pick an unmarked element a_2 and run the same process with a_2.

If a_1 has order o_1 in the multiplicative group (which is cyclic of order $q-1$), and a_2 has order o_2, we show how to construct an element a of order $o = \operatorname{lcm}(o_1, o_2)$, which generates the subgroup $\langle a_1, a_2\rangle$, as follows:

First we ensure that o_1 and o_2 are relatively prime, without changing $\operatorname{lcm}(o_1, o_2)$. For each common prime divisor p of o_1 and o_2, we determine the largest power of p dividing o_1 and o_2, respectively: $p^{f_1}|o_1$ and $p^{f_2}|o_2$. If $f_1 \le f_2$ we replace a_1 by $(a_1)^{p^{f_1}}$, dividing the order o_1 of a_1 by p^{f_1}; otherwise we proceed analogously with a_2.

After this preparation, $o = \operatorname{lcm}(o_1, o_2) = o_1 o_2$. Let g be a generator of the group $\langle a_1, a_2\rangle = \{1, g, g^2, \dots, g^{o-1}\}$. Then $\langle a_1\rangle$ is generated by $g^{o/o_1} = g^{o_2}$, and we know that $a_1 = g^{j_1 o_2}$ for some j_1. Similarly, $\langle a_2\rangle$ is generated by $g^{o/o_2} = g^{o_1}$, and $a_2 = g^{j_2 o_1}$ for some j_2.

By the extended Euclidean algorithm for calculating the greatest common divisor of o_1 and o_2, we find u_1, u_2 such that

$$o_1 u_1 + o_2 u_2 = 1.$$

The Euclidean algorithm can be carried out in $O(\log(o_1+o_2))$ iterations. We claim that $a := a_1^{u_2} a_2^{u_1}$ generates $\langle a_1, a_2\rangle$. To see this, we show that a_1 and a_2 are powers of a, in particular, $a^{o_2} = a_1$ and

$a^{o_1} = a_2$:

$$\begin{aligned} a^{o_2} = (a_1^{u_2} a_2^{u_1})^{o_2} &= g^{(j_1 o_2 u_2 + j_2 o_1 u_1) o_2} \\ &= g^{(j_1(1-o_1 u_1) + j_2 o_1 u_2) o_2} \\ &= g^{j_1 o_2 - j_1 o_1 u_1 o_2 + j_2 o_1 u_2 o_2} \\ &= g^{j_1 o_2} (g^{o_1 o_2})^{-j_1 u_1 + j_2 u_2} = a_1 \cdot 1^{-j_1 u_1 + j_2 u_2} = a_1 \end{aligned}$$

The proof of the equation $a^{o_1} = a_2$ is analogous.

In summary, the above procedure shows how we get from an element a_1 generating a subgroup $\langle a_1 \rangle$ of order o_1 and an element a_2 that is not in that subgroup to an element a generating a strictly larger subgroup $\langle a \rangle = \langle a_1, a_2 \rangle$, whose order o is a multiple of o_1. The procedure can be carried out in $O(o)$ time, assuming an array of size $q-1$ (corresponding to the whole group) is available. We repeat this procedure until we have found a generating element for the whole group. Since the size o is at least doubled in each step, the procedure can be carried out in $O(q)$ time and space in total.

6 Conclusion

In light of our NP-hardness and randomized FPT-algorithm, a natural next step is a deterministic parameterized algorithm. There are $O(n)$ local possibilities of resolving a single popular face, however, this does not immediately lead to an $O(n^k)$ time algorithm (which would place N1R in XP), since we might need to branch additionally over all possible connections between these solutions through the dual of $\mathcal{A}$, which can have an unbounded size.

References

[1] L. D. Andersen and H. Fleischner. The NP-completeness of finding A-trails in Eulerian graphs and of finding spanning trees in hypergraphs. *Discrete Applied Mathematics*, 59(3):203–214, 1995. doi:10.1016/0166-218X(93)E0172-U.

[2] K. J. Batenburg and W. A. Kosters. On the difficulty of nonograms. *ICGA Journal*, 35(4):195–205, 2012. doi:10.3233/ICG-2012-35402.

[3] S. W. Bent and U. Manber. On non-intersecting Eulerian circuits. *Discrete Applied Mathematics*, 18(1):87–94, 1987. doi:10.1016/0166-218X(87)90045-X.

[4] D. Berend, D. Pomeranz, R. Rabani, and B. Raziel. Nonograms: Combinatorial questions and algorithms. *Discrete Applied Mathematics*, 169:30–42, 2014. doi:10.1016/j.dam.2014.01.004.

[5] A. Björklund, T. Husfeld, and N. Taslaman. Shortest cycle through specified elements. In Y. Rabani, editor, *23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1747–1753, 2012. doi:10.1137/1.9781611973099.139.

[6] F. Brunck, H.-C. Chang, M. Löffler, T. Ophelders, and L. Schlipf. Reconfiguring popular faces. *Dagstuhl Reports (Seminar 22062)*, 12(2):24–34, 2022. doi:10.4230/DagRep.12.2.17.

[7] D. Bulavka, E. Colin de Verdière, and N. Fuladi. Computing Shortest Closed Curves on Non-Orientable Surfaces. In W. Mulzer and J. M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, pages 28:1–28:16, 2024. doi:10.4230/LIPIcs.SoCG.2024.28.

[8] Y. Chen and S. 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.

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

[10] P. de Nooijer, S. Nickel, A. Weinberger, Z. Masárová, T. Mchedlidze, M. Löffler, and G. Rote. Removing popular faces in curve arrangements by inserting one more curve. In E. D. Giacomo and F. Montecchiani, editors, *38th European Workshop on Computational Geometry (EuroCG 2022)*, pages 38:1–38:8, 2022. URL: https://eurocg2022.unipg.it/booklet/EuroCG2022-Booklet.pdf.

[11] P. de Nooijer, S. Terziadis, A. Weinberger, Z. Masárová, T. Mchedlidze, M. Löffler, and G. Rote. Removing popular faces in curve arrangements. In M. A. Bekos and M. Chimani, editors, *31st International Symposium on Graph Drawing and Network Visualization (GD 2023)*, pages 18–33, 2023. doi:10.1007/978-3-031-49275-4_2.

[12] H. Fleischner. *Eulerian Graphs and Related Topics, Part 1, Volume 1*, volume 45 of *Annals of Discrete Mathematics*. North-Holland, 1990.

[13] R. M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1:49–51, 1982. doi:10.1016/0167-6377(82)90044-X.

[14] J. Luo, K. D. Bowers, A. Oprea, and L. Xu. Efficient software implementations of large finite fields $GF(2^n)$ for secure storage applications. *ACM Transactions on Storage*, 8(1):2:1–2:27, 2012. doi:10.1145/2093139.2093141.

[15] J. S. Plank, K. M. Greenan, and E. L. Miller. A complete treatment of software implementations of finite field arithmetic for erasure coding applications. Technical Report UT-CS-13-717, EECS Department, University of Tennessee, 2013. URL: http://web.eecs.utk.edu/~jplank/plank/papers/UT-CS-13-717.html.

[16] J. S. Plank, E. L. Miller, K. M. Greenan, B. A. Arnold, J. A. Burnum, A. W. Disney, and A. C. McBride. GF-Complete: A comprehensive open source library for Galois field arithmetic, 2015. Revision 1.03. URL: https://github.com/ceph/gf-complete.

[17] G. Rote. The Generalized Combinatorial Lasoń–Alon–Zippel–Schwartz Nullstellensatz Lemma, 2023. arXiv:2305.10900.

[18] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

[19] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2nd edition, 2008. doi:10.1017/CBO9780511814549.

[20] D. Singmaster and J. W. Grossman. Solution to problem E2897: An Eulerian circuit with no crossings. *The American Mathematical Monthly*, 90(4):287–288, 1983. doi:10.2307/2975767.

[21] M. van de Kerkhof. Improved automatic generation of curved nonograms. Master's thesis, Utrecht University, 2017. URL: https://studenttheses.uu.nl/handle/20.500.12932/28187.

[22] M. van de Kerkhof, T. de Jong, R. Parment, M. Löffler, A. Vaxman, and M. J. van Kreveld. Design and automated generation of Japanese picture puzzles. *Computer Graphics Forum*, 38(2):343–353, 2019. doi:10.1111/cgf.13642.