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], and Günter Rote[5]

[1] Utrecht University, Utrecht, the Netherlands
m.loffler@uu.nl | t.mtsentlintze@uu.nl
[2] TU Wien, Vienna, Austria | soeren.nickel@ac.tuwien.ac.at
[3] Graz University of Technology, Graz, Austria | weinberger@ist.tugraz.at
[4] IST Austria, Maria Gugging, Austria | zuzana.masarova@ist.ac.at
[5] Freie Universität Berlin, Germany | rote@inf.fu-berlin.de

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 probabilistic FPT-approach in the number of popular faces.

Keywords: Puzzle generation · Curve arrangements · Fixed-parameter tractable (FPT).

1 Introduction

Let $\mathcal{A}$ be a set of curves which lie inside the area bounded by a 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 total intersections, which are all crossings (no tangencies).

The arrangement $\mathcal{A}$ can be seen as an embedded multigraph whose vertices are crossings of curves and whose edges are *curve segments*. $\mathcal{A}$ subdivides the

---

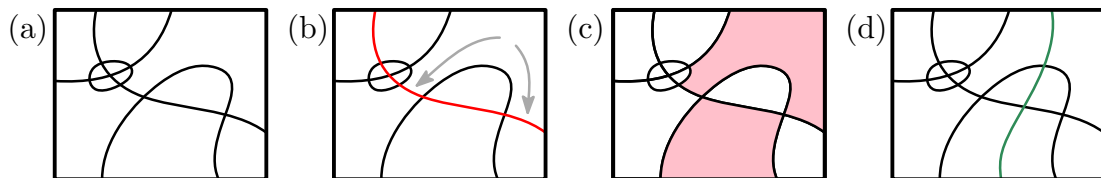[*] Authors are sorted by seniority.

Fig. 1. (a) A curve arrangement in a rectangular frame. (b) The top right face is incident to two disconnected segments of the red curve, making it *popular*. (c) All popular faces are highlighted. (d) After inserting an additional curve, no more popular faces remain.
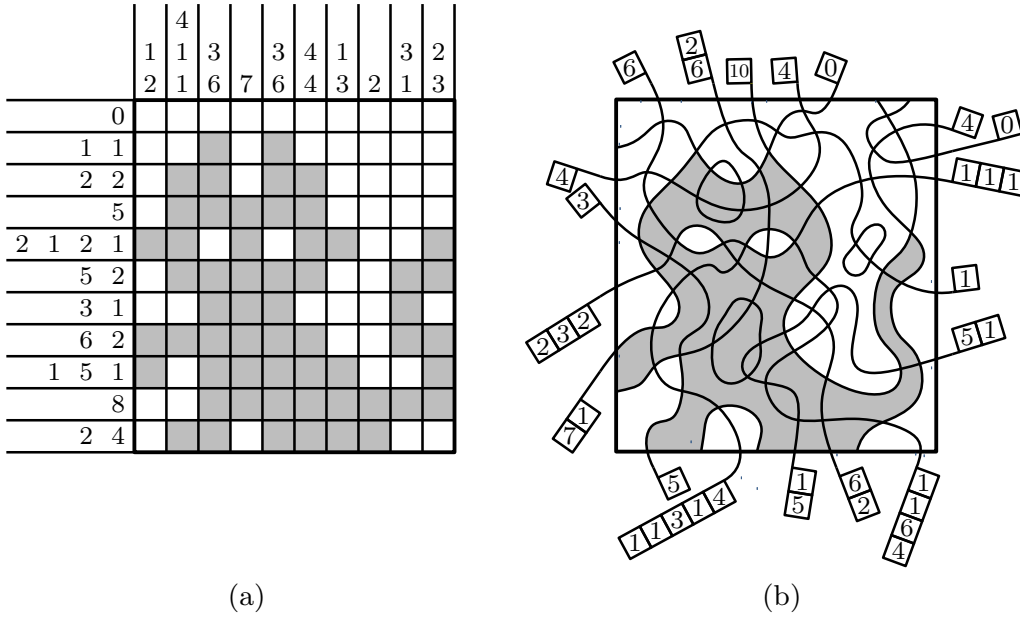
Fig. 2. Two nonogram puzzles in solved state. (a) A classic nonogram. (b) A curved nonogram.

G_{28} region bounded by the frame into *faces*. A face is *popular* when it is incident to
G_{29} multiple curve segments belonging to the same curve in $\mathcal{A}$ (see Figures 1b–c). We
G_{30} study the Nonogram 1-Resolution (N1R) problem: can one additional curve
G_{31} ℓ be inserted into $\mathcal{A}$ such that no faces of $\mathcal{A} \cup \{\ell\}$ are popular (see Figure 1d)?

G_{32} **Nonograms.** Our question is motivated by the problem of generating *curved*
G_{33} *nonograms*. Nonograms, also known as *Japanese puzzles*, *paint-by-numbers*, or
G_{34} *griddlers*, are a popular puzzle type where one is given an empty grid and a set
G_{35} of *clues* on which grid cells need to be colored. A clue consists of a sequence of
G_{36} numbers specifying the numbers of consecutive filled cells in a row or column.
G_{37} A solved nonogram typically results in a picture (see Figure 2a). There is quite
G_{38} some work in the literature on the difficulty of solving nonograms [1,3,6].
G_{39} Van de Kerkhof et al. [8] introduced *curved* nonograms, in which the puzzle
G_{40} is no longer played on a grid but on an arrangement of curves (see Figure 2b).
G_{41} In curved nonograms, clues specify numbers of filled faces of the arrangement in
G_{42} the sequence of faces incident to a common curve on one side. Van de Kerkhof et
G_{43} al. focus on heuristics to automatically generate such puzzles from a desired
G_{44} solution picture by extending curve segments to a complete curve arrangement.

G_{45} **Nonogram complexity.** Van de Kerkhof et al. observed that curved nono-
G_{46} grams come in different levels of complexity — not in terms of how hard it is to
G_{47} *solve* a puzzle, but how hard it is to understand the rules (see Figure 3). They
G_{48} state that it would be of interest to generate puzzles of a specific complexity
G_{49} level; their generators can currently do this only by trial and error.
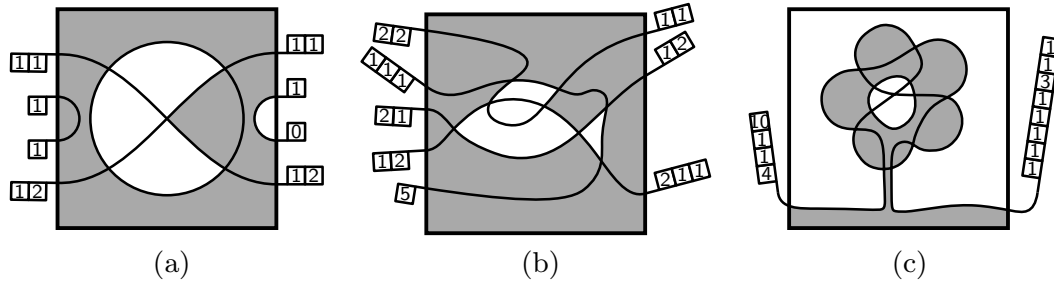
Fig. 3. Three types of curved nonograms of increasing complexity [8], shown with solutions. (a) *Basic* puzzles have no popular faces. (b) *Advanced* puzzles may have popular faces, but no self-intersections. (c) *Expert* puzzles have self-intersecting curves. We can observe closed curves (without clues) in (a) and (c).

- *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.

It is easy to see that arrangements with self-intersecting curves correspond exactly to expert puzzles. 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. [5] have investigated how popular faces in a nonogram might be removed by reconfiguring and/or reconnecting parts of curves at small local areas, which they call switches (e.g. around curve crossings), and they have proved that this problem is NP-hard. As an alternative, one may try to get rid of the popular faces by adding extra curves that cut the popular faces into smaller pieces. In this paper, we explore what we can do by inserting a single new curve into the arrangement. Clearly, inserting curves will not remove self-intersections, so we focus on changing advanced puzzles into basic puzzles; i.e., removing all popular faces.

1.1 Results.

After discussing in Section 2 how a singular face is resolved, we show in Section 3 that deciding whether we can remove all popular faces from a given curve arrangement by inserting a single curve – which we call the N1R problem – is NP-complete. However, often the number of popular faces is small, see Figure 4.
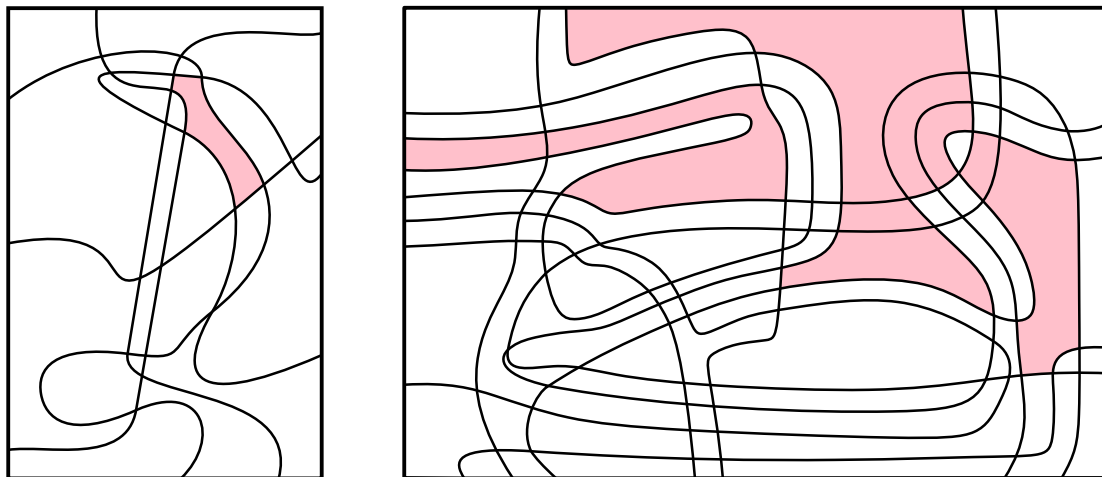
Fig. 4. Real puzzles (without clues) with all popular faces highlighted.

Hence, we are also interested in the problem parametrized by the number of popular faces k. we show in Section 4 that the problem can be solved by a randomized algorithm in FPT time.

2   Resolving one popular face by adding a single curve

As a preparation, we analyze how a single bad face F can be resolved. If F is visited three or more times by some curve, it cannot be resolved with a single additional curve ℓ, and we can immediately abort. Otherwise, there are *popular* edges among the edges of F, which belong to a curve that visits F twice. As a visual aid, we indicate each such pair of edges by connecting them with a red curve (a *curtain*), see Figure 5a or 8b.

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

1. *It visits the face F exactly once;*
2. *It does not enter or exit through a popular edge;*
3. *It separates each pair of popular edges. In other words, ℓ cuts all curtains.*

$\square$

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 if for every curtain c, the endpoints of c alternate with the vertices u and v around F, as shown in Figure 5b. This representation can be condensed as shown in Figure 5c and explained in Appendix D.

In our arguments, we will often use the dual graph $\mathcal{A}^d$ of a curve arrangement $\mathcal{A}$, where every face of $\mathcal{A}$ is represented by a vertex and edges represent faces which share a common boundary segment (not just a common crossing
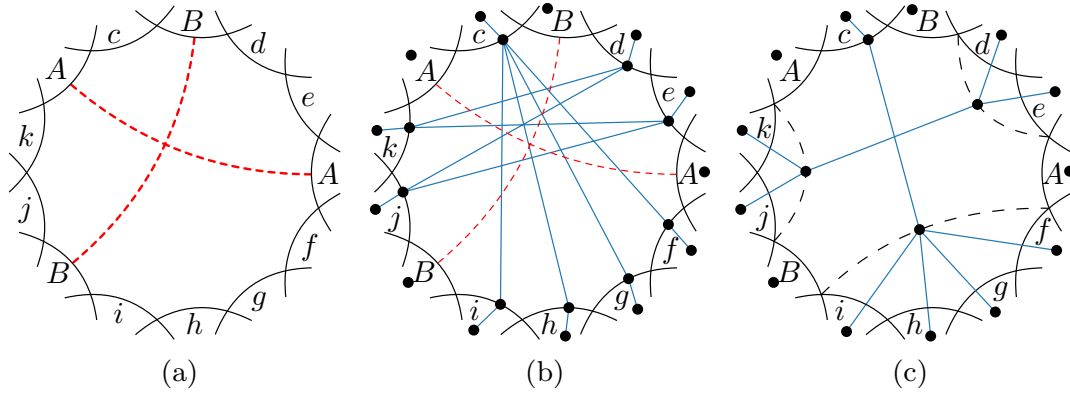
Fig. 5. Resolving a popular face F. (a) Curtains model popular edges. (b) Possible ways how ℓ can pass through F. (c) A more compact representation

point). In particular, 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 $\mathcal{A}^d$.

3   N1R is NP-complete

In order to prove NP-hardness, we reduce from *Planar Non-intersecting Eulerian Cycle*. 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 Eulerian cycles

An Eulerian cycle in a graph is a closed walk that contains every edge exactly once. An Eulerian cycle in a graph embedded into 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 Eulerian cycle 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 6a and 6b). The Eulerian cycle has to visit all of the original edges, but it does not have to cover the small vertex cycles (see Figure 6c). The following problem was proved to be NP-complete by Bent and Manber [2, Theorem 1].

Problem 1 (Planar Non-Intersecting Eulerian Cycle PNEC).   Given a planar graph embedded into the plane graph G, decide whether G contains a non-intersecting Eulerian cycle.

3.2   NP-completeness reduction

We will present a polynomial-time reduction from PNEC to N1R, i.e., we will create a curve arrangement $\mathcal{A}$ containing popular faces based on a planar input graph G of PNEC, s.t. there exists a curve ℓ for which $\mathcal{A} \cup \ell$ contains no popular
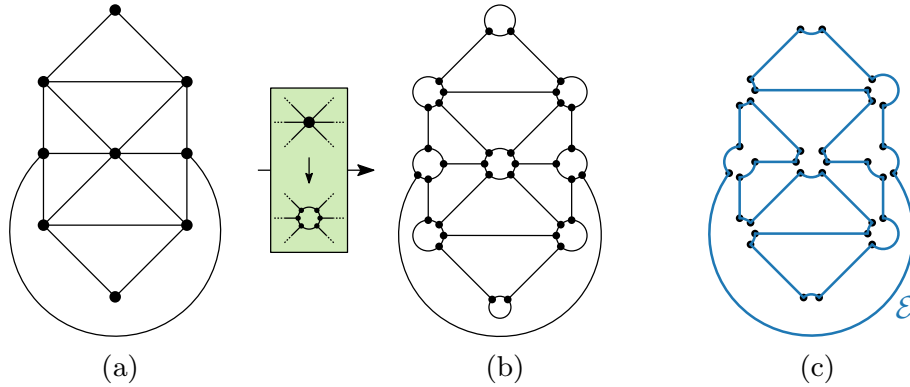
Fig. 6. Vertices of the graph G (a) are replaced by cycles (b). A non-intersecting cycle drawn on the modified graph visiting all original edges (c).

G_{126} faces if and only if G contains a non-intersecting Eulerian cycle. We assume that
G_{127} G is 2-edge-connected and all vertices have even degree, because otherwise, G
G_{128} clearly cannot contain an Eulerian cycle. We also replace every self-loop with a
G_{129} path of length two, without affecting the existence of a planar non-intersecting
G_{130} Eulerian cycle.
G_{131} The reduction is gadget based. We will represent every vertex $v \in V$ with a
G_{132} vertex gadget $\mathcal{N}(v)$ and every edge $e = (u, v) \in E$ with an edge gadget $\mathcal{L}(e)$ or
G_{133} $\mathcal{L}(u, v)$. Both gadgets are sets of curves starting and ending at the frame, and
G_{134} $\mathcal{A} = \bigcup_{v \in V} \mathcal{N}(v) \cup \bigcup_{e \in E} \mathcal{L}(e)$.

G_{135} **Vertex gadgets.** The vertex gadgets consist of curves in one of three basic
G_{136} shapes shown in Figure 7a, which we call beakers. We place one beaker per
G_{137} incident edge of v, at the position of v, all rotated, s.t. their *bases* (the lower
G_{138} ends in Figure 7a) overlap in a specific pattern. The *opening* of each beaker (the
G_{139} upper ends in Figure 7a) will point outwards. We use three variants of the vertex
G_{140} gadget, depending on the vertex degree.
G_{141} The vertex gadget for a degree-two vertex is simply made up of two overlaying
G_{142} Type-I beakers (see Figure 7b). Since ℓ must cross the two curtains c_1 and c_2, it
G_{143} must connect the two points p_1, p_2 by crossing the overlap of the two beakers (a
G_{144} face of degree two, marked in green in Figure 7b). Since by Observation 1, ℓ can
G_{145} enter any beaker only once, the routing of ℓ as shown in Figure 7c is forced and
G_{146} corresponds exactly to the traversal of a planar non-intersecting Eulerian cycle
G_{147} through a vertex of degree two.
G_{148} The vertex gadget $\mathcal{N}(v)$ for a degree-four vertex v consists of four Type-I
G_{149} beakers, one per incident edge, which form the intersection pattern of Figure 7d.
G_{150} Since ℓ must cross the four curtains, it must enter or exit the gadget at least four
G_{151} times through the thick blue edges in Figure 7e and the vertices p_1, p_2, p_3, p_4 of
G_{152} the dual graph $\mathcal{A}^d$. Since ℓ cannot cross itself, there are only two possibilities
G_{153} how ℓ can pass through $\mathcal{N}(v)$: It can connect p_1 with p_2 and p_3 with p_4, as in
G_{154} Figure 7f, or p_1 with p_4 and p_2 with p_3. Both possibilities can be realized by
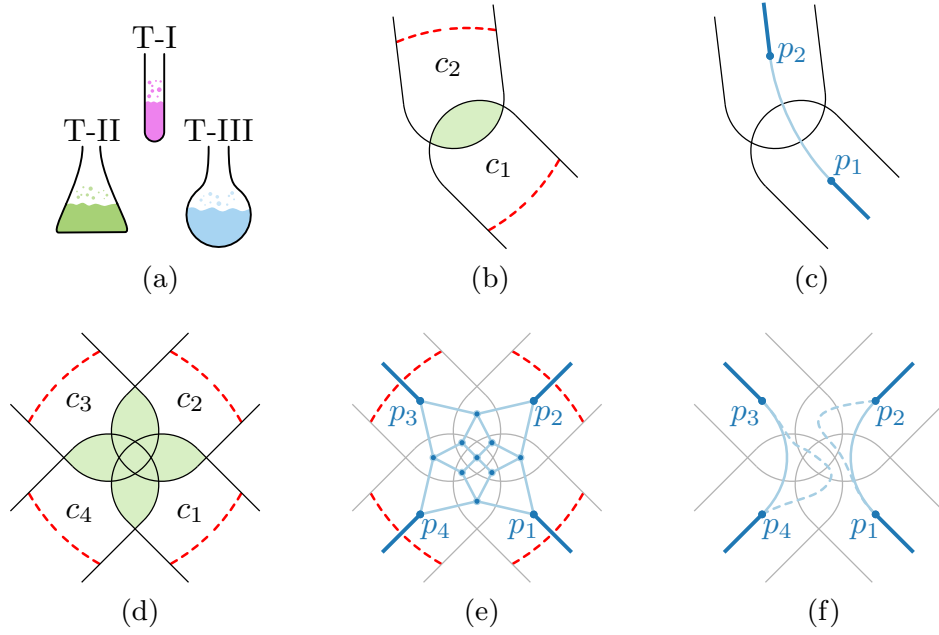
Fig. 7. (a) Basic beaker curve shapes. (b) Degree two gadget (2 Type-I beakers) and (c) its forced resolution. (d) Degree four gadget (4 Type-I beakers). (e) Dual graph of the degree 4 gadget. (f) A possible curve ℓ in light blue; some alternative routings, which connect the same endpoints, in dashed light blue.

routings of ℓ, and they correspond precisely to the ways how a non-intersecting Eulerian cycle can pass through the edges incident to v. Note that the exact routing of ℓ can vary inside $\mathcal{N}(v)$ (indicated by the dashed lines in Figure 7).

The vertex gadget for a vertex v of degree $d \geq 6$ is more complex. We place $d - 1$ Type-II beakers $c_1, \ldots, c_{d-1}$ symmetrically around the location of v (Figure 8a). Each beaker intersects four adjacent beakers (two on each side), with the exception that $c_{d/2-1}$ and $c_{d/2+1}$ (dark green curves in Figure 8a) do not intersect. We place an additional Type-III beaker c_d (the light green curve in Figure 8a) that surrounds all bases of the Type-II beakers and protrudes between c_{d-1} and c_1, such that the intersection pattern of Figure 8a arises.

All popular faces and curtains in $\mathcal{N}(v)$ are shown in Figure 8b. The dual of the construction is shown in Figure 8c. The curtains in the green faces force ℓ to pass from these faces to the adjacent small faces with the blue boundaries. This constrains ℓ to pass through a chain of faces as shown in Figure 8d. The passages from these faces to other neighboring faces can now be excluded, and the corresponding edges have been removed from the dual graph in Figure 8d.

The curtains in the openings of the beakers force the outer blue endpoints of ℓ. The endpoint in beaker c_i will be called p_i. Now we analyze which of these endpoints can be connected with each other. We see that in most cases, p_i can only be connected to p_{i-1} or p_{i+1} without going through another endpoint. The exception is $p_{d/2-1}$ and $p_{d/2+1}$, which can be connected via the inner loop from q_1 to q_2. However, this connection would cut off $p_{d/2}$ from the remaining
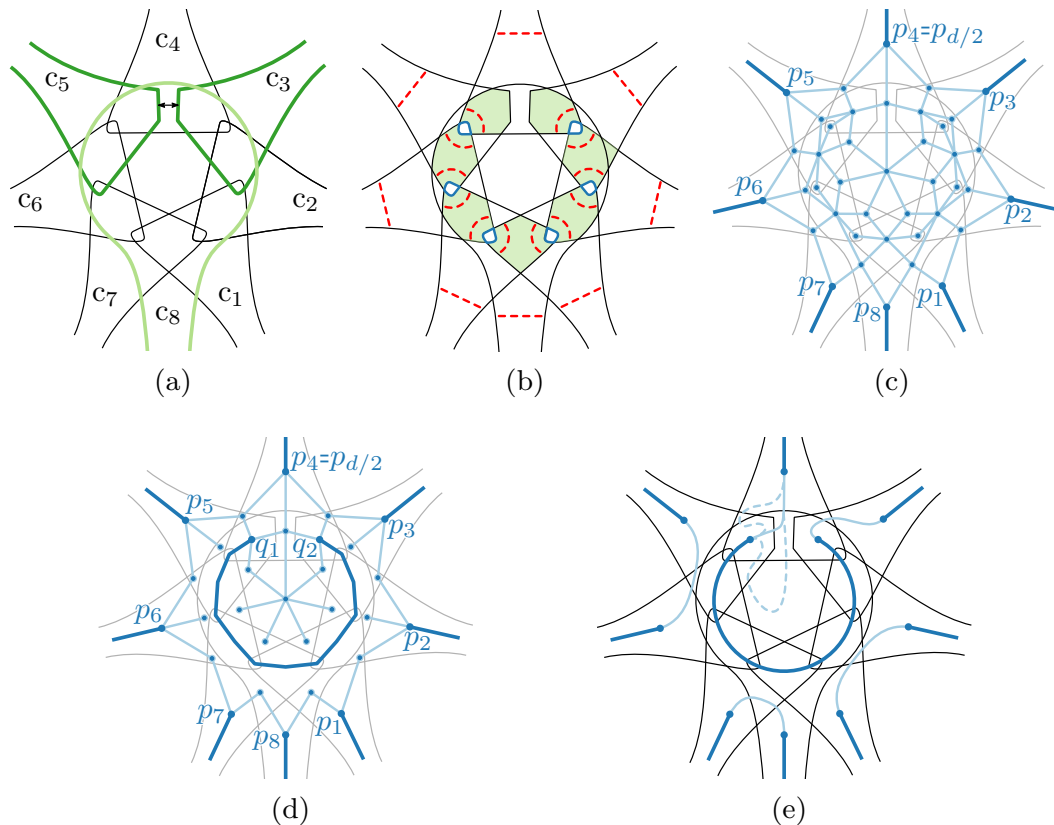
Fig. 8. (a) Vertex gadget $\mathcal{N}(v)$ for a degree-8 vertex v, (b) its curtains and (c) dual graph. (d) Highlighted light green faces in (b) force the dark blue connections in the dual graph and restrict it. (e) One of two symmetric possibilities for the splitting curve ℓ. The dashed lines show different possible routings of ℓ.

G_{177} points. We conclude that the visits of ℓ to $\mathcal{N}(u)$ must match endpoints p_i that
G_{178} are adjacent in the circular order. There are two matchings, which correspond
G_{179} to the two possibilities how a non-intersecting Eulerian cycle can visit v. Both
G_{180} possibilities can be realized by routings of ℓ; one is shown in Figure 8e, and the
G_{181} other is symmetric.

G_{182} We now have placed vertex gadgets for all vertices. They require ℓ to connect
G_{183} to an endpoint in each opening of a beaker. With these openings, we will now
G_{184} construct the edge gadgets.

G_{185} **Edge gadgets.** Let $e = (u, v) \in E$ be an edge in G. Then there are two vertex
G_{186} gadgets $\mathcal{N}(u)$ and $\mathcal{N}(v)$ already placed. In particular, we placed one beaker in
G_{187} the gadgets per incident edge at u or v, i.e. two per edge.

G_{188} We now elongate the open ends of these beakers and route them along the
G_{189} edge e according to the embedding of G given in the input (recall that G is a plane
G_{190} graph) until they almost meet at the center point of e. We bend the ends of each
G_{191} beaker outward, routing them into the two faces incident to e. Additionally, we
G_{192} place two more curves on top (shown in green), forming the intersection pattern

Fig. 9. Edge gadget $\mathcal{L}(u, v)$ connecting two beakers from $\mathcal{N}(u)$ and $\mathcal{N}(v)$ with two additional curves. Open ends of all curves are collected into two bundles of parallel curves that lead into the incident faces. The inside of the two beakers are connected via a chain of popular faces in $\mathcal{L}(u, v)$ (shaded in light green). Other bundles, like the two groups of four light blue curves in the left half, can freely cross either beaker.
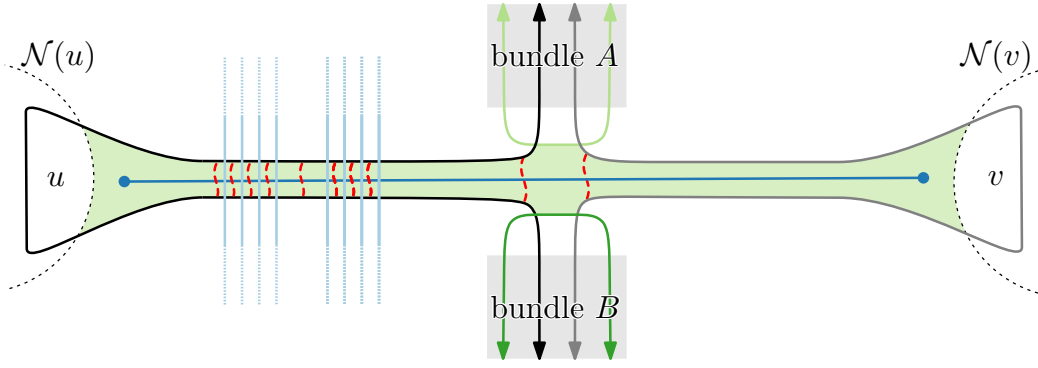
of Figure 9. This results in two *bundles A* and *B*, each consisting of four parallel curves. (The light blue curves in the left half are not part of the gadget; they are two bundles that come from other gadgets.)

This connects a popular face in $\mathcal{N}(u)$ to one in $\mathcal{N}(v)$, forming one big popular face in $\mathcal{L}(u, v)$. An arbitrary number of curves may cross the opening of a beaker. The face is then simply divided into a chain of consecutive popular faces. In each of these faces, except the left- and right-most faces, which contain the blue endpoints, ℓ has to leave through two specific edges in order to cut the curtains. This forces ℓ to pass straight through $\mathcal{L}(u, v)$ from $\mathcal{N}(u)$ to $\mathcal{N}(v)$ along the thin dark-blue horizontal axis.

It remains to describe how the open ends of the curves in the bundles are routed to the frame (since all curves other than ℓ have start and end at the frame). This is not difficult because these bundles can cross quite freely without creating popular faces. Each bundle consists of a unique set of curves, except for the two bundles from one edge gadget. A bundle that originates from the edge gadget $\mathcal{L}(e)$ can thus cross any bundle from a different edge gadget without creating popular faces. It can cross a different edge gadget $\mathcal{L}(e')$ by passing over one of its beakers, as shown with the light-blue curves.

Since G does not contain self-loops, we route each bundle along a path in the dual of G to the outer face of G, and then connect it to the frame, see Figure 10. A popular face might only be created when a bundle crosses the other bundle from the same edge gadget, which can be avoided by routing them in parallel.

The curves in a bundle run in parallel. Two bundles originating from an edge gadget $\mathcal{L}(e)$ have different curves as their outside curves. Hence, no popular faces are created between two bundles, and we can make the following statement.

Observation 2 *All popular faces in $\mathcal{A}$ are contained in vertex gadgets and edge gadgets (the faces with dashed red curtains in Figures 7b, 7d, 8b, and 9).*
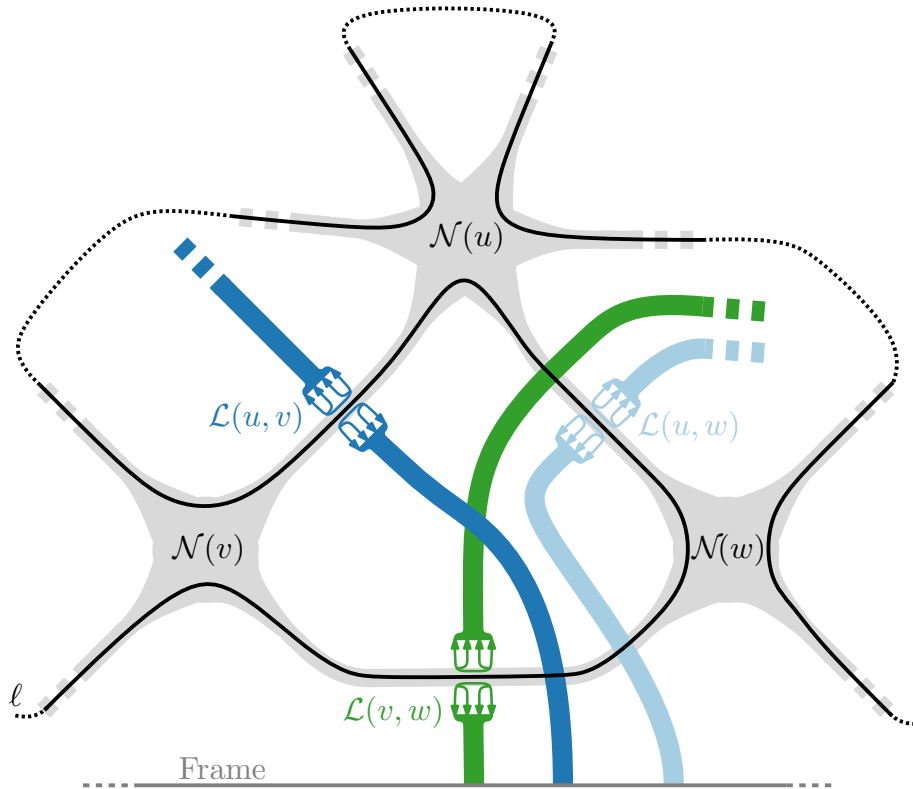
Fig. 10. Schematic representation of three vertex and edge gadgets. The bundles are routed through beakers of other edges ending at the frame (partially shown at the bottom of the figure). A possible routing of ℓ is shown with a black curve.

The next theorem follow from the construction and the resulting correspondence between resolving curves and non-crossing Eulerian cycles. The proof can be found in Appendix A.

Theorem 1. *N1R is* NP-*complete.*

Adaption to open curves.   The reduction assumes that ℓ is a closed loop. It can be adapted to work for open curves, i.e., we can create the arrangement $\mathcal{A}$, for which there exists an open curve ℓ' starting and ending at the frame, s.t. $\mathcal{A} \cup \ell'$ does not contain any popular faces, if and only if G contains a planar non-intersecting Eulerian cycle $\mathcal{E}(G)$.

The reduction creates $\mathcal{A}$ in the same fashion as above, except that we do not add the edge gadget for exactly one edge $e_o = (v_o, u_o)$ on the outer face of G. Instead, the open curves of the openings of the beakers in $\mathcal{N}(u_o)$ and $\mathcal{N}(v_o)$, which would normally form the edge gadget $\mathcal{L}(u_o, v_o)$ are simply connected to the frame. This forces ℓ' to start (and end) at the frame between the points at which the beakers connect to the frame, in order to properly split the popular face in the opening of these beakers. It is now easy to see that all other properties still hold and N1R remains NP-complete even when ℓ' can be an open curve.
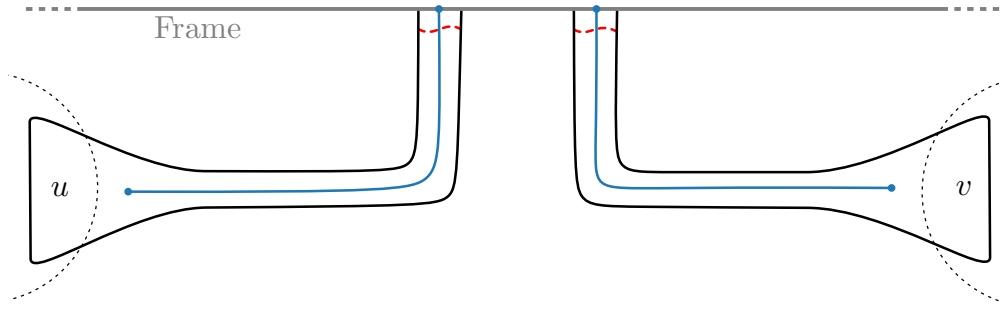
Fig. 11. By routing both ends of an open beaker in parallel to the frame, we force ℓ to start (or end) at the frame between the two connection points of the beaker.

G_{233}

4   Randomized **FPT**-algorithm for **N1R**

G_{234} In this section, we show that N1R with k popular faces can be solved by a ran-
G_{235} domized algorithm in $O\left(2^k\mathrm{poly}(n)\right)$ time, placing N1R in the class randomized
G_{236} 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
G_{237} modified dual graph G, subject to a constraint that certain edges must be visited.

G_{238} *Problem 2 (Simple Cycle with Edge Set Constraints SNESC).* Given an undi-
G_{239} rected graph $G = (V, E)$ and k subsets $S_1, S_2, \ldots, S_k \subseteq E$ of edges, find a simple
G_{240} cycle, if it exists, that contains exactly one edge from each set S_i.

G_{241} We start with the dual graph of the given curve arrangement $\mathcal{A}$. We replace
G_{242} the vertex corresponding to the i-th popular face f with a set S_i of edges mod-
G_{243} eling the ways how an additional curve can cut all curtains of f, as described
G_{244} in Section 2 and shown in Figure 5b. To be specific, we place a vertex on each
G_{245} curve segment s bounding f and connect it to the vertex of the face that is
G_{246} adjacent to f across s. Further we connect two such vertices on curve segments
G_{247} if a curve entering f through one segment and exiting through the other would
G_{248} cut all curtains of f. The latter connecting edges, which run through f, form
G_{249} the set S_i. There is a one-to-one correspondence between the simple cycles
G_{250} containing exactly one edge of every set S_i and the resolving curves for $\mathcal{A}$.
G_{251} We will describe a randomized algorithm for Problem 2, extending an algo-
G_{252} rithm of Björklund, Husfeld, and Taslaman [4].

G_{253} **Theorem 2.** (a) *The SNESC problem on a graph with n vertices and $m \geq$*
G_{254} *n edges can be solved in $O\left(2^k mn^2 \log \frac{2m}{n} \cdot |V(S_1)| \cdot W\right)$ time and $O(2^k n + m)$*
G_{255} *space with a randomized Monte-Carlo algorithm, with probability at least*
G_{256} *$1 - 1/n^W$, for any $W \geq 1$. Here, $V(S_1)$ denotes the set of vertices of the edges*
G_{257} *in S_1 (note that S_1 can be chosen to be the smallest set among $S_1, \ldots, S_k$).*
G_{258} *The model of computation is the Word-RAM with words of size $\Theta(k + \log n)$.*
G_{259} (b) *There is an alternative algorithm (described in Appendix B.3) needing only*
G_{260} *polynomial space, namely $O(kn + m)$, at the expense of an additional factor*
G_{261} *k in the runtime. It uses words of size $\Theta(\log n)$.*

$G262$ *Both algorithms find the cycle with the smallest number of edges if it exists*
$G263$ *(with high probability).*

$G264$ If $\mathcal{A}$ has n faces, the graph G has $O(n)$ vertices and $m = O(n^2)$ edges. The
$G265$ quadratic blow-up of m results from the construction as shown in Figure 5b. The
$G266$ number of edges can be reduced to $O(n)$, as shown in Figure 5c and discussed
$G267$ in Appendix D. The number k of popular faces is the same as the number k of
$G268$ edge sets S_i.

$G269$ With the alternative algorithm with polynomial space, since $k \leq n$, we get:

$G270$ **Corollary 1.** *The N1R problem with k popular faces in a curve arrangement*
$G271$ *with n faces can be solved in expected time $O(2^k \mathrm{poly}(n))$ and $O(kn)$ space.* $\square$

$G272$ We first give a high-level overview of the algorithm. We start by assigning
$G273$ random weights to the edges from a sufficiently large finite field $\mathbb{F}_q$ of charac-
$G274$ teristic 2. Such a field exists for every size q that is a power of 2. In a field of
$G275$ characteristic 2, the law $x + x = 0$ holds, and therefore terms cancel when they
$G276$ occur an even number of times. The weight of a *walk* (with vertex and edge
$G277$ repetitions allowed) is obtained by multiplying the edge weights of all visited
$G278$ edges. Our goal is now to compute the sum of weights all closed walks, of given
$G279$ length, that satisfy the edge set constraints. The characteristic-2 property will
$G280$ ensure that the unwanted walks, those which are not simple, cancel, while a
$G281$ simple closed walk makes a nonzero contribution and leads to a nonzero sum
$G282$ with high property. The crucial idea is that, while these sets of closed walks can
$G283$ be very complicated, we can compute the aggregated sum of their weights in
$G284$ polynomial time. We have to anchor these walks at some starting vertex b, and
$G285$ we choose b to be one of the vertices incident to an edge of S_1.

$G286$ More precisely, for each such vertex b, and for increasing lengths $l = 1, 2, \ldots, n$,
$G287$ the algorithm computes the quantity $\hat{T}_b(l)$, which is the sum of the weights of
$G288$ all closed walks that

$G289$ $-$ start and end at b,
$G290$ $-$ have their first edge in S_1,
$G291$ $-$ use exactly one edge from each set S_i (and use it only once),
$G292$ $-$ and consist of l edges.

$G293$ We consider the edge weights as variables and regard $\hat{T}_b(l)$ as a function
$G294$ of these variables. The result is a polynomial where each term is a product of
$G295$ 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
$G296$ lemma, which is a straightforward adaptation of a lemma of Björklund et al. [4].

$G297$ **Lemma 1.** *a) Suppose there exists a simple cycle of length l that satisfies the*
$G298$ *edge set constraints and that goes through an edge of S_1 incident to b. Then*
$G299$ *the polynomial $\hat{T}_b(l)$ is homogeneous of degree l and is not identically zero.*
$G300$ *b) If there is no such cycle of length $\leq l$, the polynomial $\hat{T}_b(l)$ is identically zero.*

G_{301} The lemma is based on the fact that each term in the polynomial $\hat{T}_b(l)$
G_{302} represents some closed walk. A term coming from a walk that visits a vertex
G_{303} twice can be matched with another walk, which traverses a loop in the opposite
G_{304} direction and contributes the same term. Since the field has characteristic 2,
G_{305} these terms cancel. A term coming from a simple walk does not cancel. The
G_{306} proof of Lemma 1 is given in Appendix B.5.

G_{307} In case (a) of Lemma 1, it follows from the Schwartz-Zippel Lemma [15,
G_{308} Corollary 1] (see also [14, Corollary Q1]) that, for randomly chosen weights in $\mathbb{F}_q$,
G_{309} $\hat{T}_b(l)$ is nonzero with probability at least $1 - \text{degree}/|\mathbb{F}_q| = 1 - l/q \geq 1 - n/q$.

G_{310} Thus, if we choose $q > n^2$, we have a success probability of at least $1 - 1/n$ for
G_{311} finding the shortest cycle when we evaluate the quantities $\hat{T}_b(l)$ for increasing l
G_{312} until they become nonzero. The success probability can be boosted by repeating
G_{313} the experiment with new random weights.

G_{314} In the unlikely case of a failure, the algorithm may err by not finding a
G_{315} solution although a solution exists, or by finding a solution that is not shortest.
G_{316} The last possibility is not an issue for our original problem, where we just ask
G_{317} about the existence of a cycle, of arbitrary length.

G_{318} In the following, we will discuss how we can compute the quantities $\hat{T}_b(l)$,
G_{319} and the runtime and space requirement for this calculation. We describe the
G_{320} method for actually recovering the cycle after we have found a nonzero value in
G_{321} Appendix B.4.

G_{322} ## 4.1   Computing sums of path weights by dynamic programming

G_{323} We cannot compute the desired sums $\hat{T}_b(l)$ directly, but have to do this incre-
G_{324} mentally via a larger variety of quantities $T_b(R, l, v)$ that are defined as follows:
G_{325} 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
G_{326} the sum of the weights of all walks that
G_{327} $-$ start at b,
G_{328} $-$ have their first edge in S_1,
G_{329} $-$ end at v,
G_{330} $-$ consist of l edges,
G_{331} $-$ use exactly one edge from each set S_i with $i \in R$ (and use it only once),
G_{332} $-$ contain no edge from the sets S_i with $i \notin R$.
G_{333} The walks that we consider here differ from the walks in $\hat{T}_b(l)$ in two respects:
G_{334} They end at a specified vertex v, and the set R keeps track of the sets S_i that
G_{335} were already visited. The quantities $\hat{T}_b(l)$ that we are interested in arise as a
G_{336} special case when we have visited the full range $R = \{1, \ldots, k\}$ of sets S_i and
G_{337} arrive at $v = b$:
G_{338}
$$\hat{T}_b(l) = T_b(\{1, \ldots, k\}, l, b).$$

G_{339} We compute the values $T_b(R, l, v)$ for increasing values $l = 1, \ldots, n$. The
G_{340} starting values for $l = 1$ are straightforward from the definition.
G_{341} To compute $T_b(R, l, v)$ for $l \geq 2$, we collect all stored values of the form
G_{342} $T_b(R', l - 1, u)$ where (u, v) is an edge of G and R' is derived from R by taking
G_{343} into account the sets S_i to which (u, v) belongs. We multiply these values with
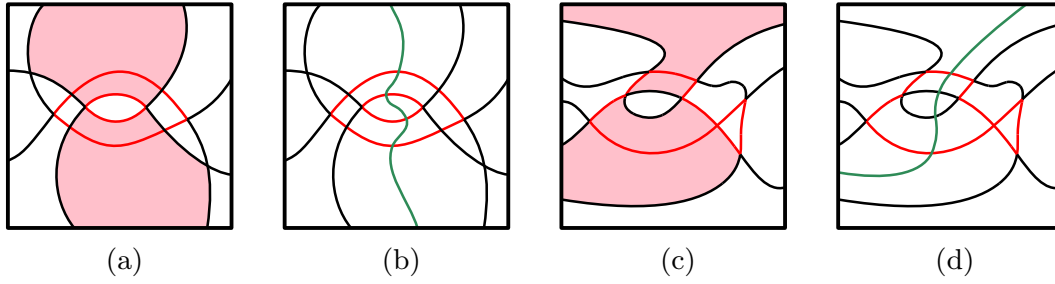
Fig. 12. Input (a,c) and resulting output (b,d) generated by the implementation; the green curve resolves the popular faces with the smallest number of crossings.

the edge weight w_{uv} and sum them up. If (u,v) is in some S_i but $i \notin R$, we don't 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

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

4.2   Runtime and space

The finite field additions and multiplications in (1) take constant time, see Appendix B.2. Similarly, the set operation $R \setminus I(u,v)$ on subsets of $\{1,\dots,k\}$ 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 (1) 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 Lemma 1, 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 W times, reducing the failure probability to $1/n^W$.

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 Appendix B.4.

Figure 12 shows initial results of an implementation of our algorithm on two small test instances; see also [10].

5   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 $O(n^k)$ 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. Batenburg, K.J., Kosters, W.A.: On the difficulty of nonograms. ICGA Journal **35**(4), 195–205 (2012). https://doi.org/10.3233/ICG-2012-35402
2. Bent, S.W., Manber, U.: On non-intersecting Eulerian circuits. Discrete Applied Mathematics **18**(1), 87–94 (1987). https://doi.org/10.1016/0166-218X(87)90045-X
3. Berend, D., Pomeranz, D., Rabani, R., Raziel, B.: Nonograms: Combinatorial questions and algorithms. Discrete Applied Mathematics **169**, 30–42 (2014). https://doi.org/10.1016/j.dam.2014.01.004
4. Björklund, A., Husfeld, T., Taslaman, N.: Shortest cycle through specified elements. In: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1747–1753. SODA '12, Society for Industrial and Applied Mathematics, USA (2012). https://doi.org/2095116.2095255
5. Brunck, F., Chang, H.C., Löffler, M., Ophelders, T., Schlipf, L.: Reconfiguring popular faces. Dagstuhl Reports (Seminar 22062) **12**(2), 24–34 (2022). https://doi.org/10.4230/DagRep.12.2.17
6. Chen, Y., Lin, S.: A fast nonogram solver that won the TAAI 2017 and ICGA 2018 tournaments. ICGA Journal **41**(1), 2–14 (2019). https://doi.org/10.3233/ICG-190097
7. Karp, R.M.: Dynamic programming meets the principle of inclusion and exclusion. Oper. Res. Lett. **1**, 49–51 (1982). https://doi.org/10.1016/0167-6377(82)90044-X
8. van de Kerkhof, M., de Jong, T., Parment, R., Löffler, M., Vaxman, A., van Kreveld, M.J.: Design and automated generation of Japanese picture puzzles. Comput. Graph. Forum **38**(2), 343–353 (2019). https://doi.org/10.1111/cgf.13642
9. Luo, J., Bowers, K.D., Oprea, A., Xu, L.: Efficient software implementations of large finite fields $GF(2^n)$ for secure storage applications. ACM Trans. Storage **8**(Article 2), 2:1–2:27 (2012). https://doi.org/10.1145/2093139.2093141
10. de Nooijer, P.: Resolving Popular Faces in Curve Arrangements. Master's thesis, Utrecht University (2022), https://studenttheses.uu.nl/handle/20.500.12932/494
11. de Nooijer, P., Nickel, S., Weinberger, A., Masárová, Z., Mchedlidze, T., Löffler, M., Rote, G.: Removing popular faces in curve arrangements by inserting one more curve. In: Giacomo, E.D., Montecchiani, F. (eds.) Abstracts of the 38th European Workshop on Computational Geometry (EuroCG 2022). pp. 38:1–38:8 (Mar 2022), https://eurocg2022.unipg.it/booklet/EuroCG2022-Booklet.pdf
12. Plank, J.S., Greenan, K.M., Miller, E.L.: A complete treatment of software implementations of finite field arithmetic for erasure coding applications. Tech. Rep. UT-CS-13-717, EECS Department, University of Tennessee (Oct 2013), http://web.eecs.utk.edu/~jplank/plank/papers/UT-CS-13-717.html

13. Plank, J.S., Miller, E.L., Greenan, K.M., Arnold, B.A., Burnum, J.A., Disney, A.W., McBride, A.C.: GF-Complete: A comprehensive open source library for Galois field arithmetic (Jan 2015), https://github.com/ceph/gf-complete, revision 1.03

14. Rote, G.: The Generalized Combinatorial Lasoń–Alon–Zippel–Schwartz Nullstellensatz Lemma (2023), arXiv:2305.10900 [math.CO]

15. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4), 701–717 (1980). https://doi.org/10.1145/322217.322225

16. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press, 2nd edn. (2008). https://doi.org/10.1017/CBO9781139165464

17. Wahlström, M.: Abusing the Tutte matrix: An algebraic instance compression for the K-set-cycle problem. In: 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013). Leibniz International Proceedings in Informatics (LIPIcs), vol. 20, pp. 341–352. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2013). https://doi.org/10.4230/LIPIcs.STACS.2013.341

A   Proof of Theorem 1

G_{431}

G_{432} **Theorem 1** *N1R is* NP-*complete.*

G_{433} *Proof.* Assume we are given a non-intersecting Eulerian cycle $\mathcal{E}(G)$ of G as a
G_{434} permutation of the edges. We now show how to construct the curve ℓ. We choose
G_{435} a random edge $e = (u, v)$ in the permutation and start drawing ℓ at the endpoint
G_{436} inside the beaker of $\mathcal{L}(u, v)$ originating from $\mathcal{N}(v)$ along the thin blue axis cross-
G_{437} ing all popular faces as described above.This will end at an open endpoint inside
G_{438} $\mathcal{N}(u)$. At this point we either connect to the left or right endpoint according
G_{439} to the next edge in $\mathcal{E}(G)$ (which is possible, since $\mathcal{E}(G)$ is non-intersecting). We
G_{440} make this connection either directly or via the forced inner circular part of ℓ in
G_{441} $\mathcal{N}(u)$ if one of the two involved endpoints is in beaker $c_{d(u)/2}$ (where $d(u)$ is the
G_{442} degree of u). These connections are made as described above.Now we are again
G_{443} at an endpoint in a beaker. We can repeat this procedure until we cycle back to
G_{444} e, at which point we will have reconnected to our starting point. Since $\mathcal{E}(G)$ by
G_{445} definition visits every edge exactly once, before cycling back to the first edge, we
G_{446} know that every popular face in the edge gadgets is split. Moreover, since $\mathcal{E}(G)$
G_{447} visits every vertex v exactly $\delta(v)/2$ times and $\mathcal{E}(G)$ is non-intersecting, we know
G_{448} that one of the two possible ways of connecting all endpoints in $\mathcal{N}(v)$ can be
G_{449} chosen to resolve all popular faces in $\mathcal{N}(v)$ (and all other vertex gadgets). Since
G_{450} by Observation 2 there are no other popular faces in $\mathcal{A}$, $\mathcal{A} \cup \ell$ does not contain
G_{451} any popular faces.

G_{452} Now assume we are given a curve ℓ, s.t. $\mathcal{A} \cup \ell$ does not contain any popular
G_{453} faces. The order, in which ℓ traverses all edge gadgets gives us a permutation of
G_{454} all edges. Since both variants of resolving all popular faces in a vertex gadget
G_{455} connect any endpoint only to an endpoint in a neighboring beaker, consecutive
G_{456} edges in this permutation share an endpoint and both belong to the same face
G_{457} and this permutation is a non-intersecting Eulerian cycle. We have shown that
G_{458} there exists a curve ℓ, s.t., $\mathcal{A} \cup \ell$ contains no popular faces, if and only if G
G_{459} contains a non-intersecting Eulerian cycle and therefore N1R is NP-hard.

G_{460} It is easy to see that N1R is in NP. The input arrangement can be rep-
G_{461} resented as a plane graph in which the edges are marked as belonging to the
G_{462} different curves or to the frame boundary. The certificate is an extension of this
G_{463} arrangement by a resolving curve ℓ. Since ℓ cannot visit a face more than once,
G_{464} the certificate is of polynomial size. It can be easily verified in polynomial time
G_{465} whether it is valid, in particular, whether it contains no popular faces.   □

B   Additional details of the randomized **FPT**-algorithm

G_{466}

B.1   Assumption on the word length

G_{467} Our algorithm needs to store a table with $\Theta(2^k n)$ entries, namely the entries
G_{468} $T_b(R, l, v)$ with fixed b and l. We assume that these fit in memory and can be
G_{469} addressed in constant time. Thus, it is reasonable to assume a word length of at
G_{470} least $k + \log n$ bits.

G_{471}

B.2   Finite field calculations

G_{472}
G_{473}

Each evaluation of the recursion (1) involves additions and multiplications in the finite field $\mathbb{F}_q$, where $q = 2^s > n$ is a power of 2.

G_{474}
G_{475}
G_{476}
G_{477}

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.

G_{478}
G_{479}
G_{480}
G_{481}

For implementing the algorithm in practice, arithmetic in $\mathbb{F}_{2^s}$ is supported by a variety of powerful libraries. see for example [13]. The size that these libraries conveniently offer (e.g., $q = 2^{32}$) will be mostly sufficient for a satisfactory success probability.

G_{482}
G_{483}
G_{484}

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 W times.

G_{485}
G_{486}
G_{487}
G_{488}
G_{489}
G_{490}
G_{491}
G_{492}

We will describe 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 [9] or Plank, Greenan, and Miller [12]. 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.

G_{493}
G_{494}
G_{495}

We propose to use the split table method, which is simple and implements multiplication by a few look-ups in small precomputed tables, see [9, Section 3.3.1] or [12, Section 6.5].

G_{496}

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

G_{497}
G_{498}
G_{499}
G_{500}

According to [16, 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 [16] points out that this bound is not tight; there are also faster methods.)

G_{501}
G_{502}
G_{503}
G_{504}

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, by elementwise multiplication of the two polynomials, and reducing the product by successive elimination of terms of degree larger than d.

G_{505}
G_{506}

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

G_{507}

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

G_{508}
G_{509}
G_{510}
G_{511}
G_{512}
G_{513}

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

G_{514}
G_{515}

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.

G_{516}
G_{517}
G_{518}
G_{519}
G_{520}

Thus, after some initial overhead of $O(n \log^2 n)$ time, which is negligible in the context of the overall algorithm, with $O(n)$ space, arithmetic in $\mathbb{F}_q$ can be carried out in $O(1)$ time in the Word-RAM model, where arithmetic, logic, and addressing operations on words with $\Theta(\log n)$ bits are considered as constant-time operations.

G_{521}
G_{522}
G_{523}

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 Appendix C.

G_{524}

B.3   Reduction to polynomial space

G_{525}
G_{526}
G_{527}
G_{528}
G_{529}
G_{530}

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 [7] for the Traveling Salesman Problem. The possibility of applying this trick in the context of our problem was already mentioned in Björklund et al. [4].

G_{531}
G_{532}

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

G_{533}
G_{534}
G_{535}
G_{536}
G_{537}
G_{538}
G_{539}
G_{540}

− 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.

G_{541}
G_{542}
G_{543}
G_{544}

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 (1), we have to keep track of the number j of edges from $S_1 \cup \cdots \cup S_k$.

G_{545}
G_{546}
G_{547}
G_{548}

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:

G_{549}

Lemma 2.

G_{550}

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

G_{551}
G_{552}
G_{553}
G_{554}

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; thus, every set S_i is visited *exactly* once.   □

G_{555} The sign $(-1)^{|F|}$ is of course irrelevant over a field of characteristic 2.

G_{556} To reduce the space requirement as much as possible, we organize the com-
G_{557} putation as follows. First, each starting point b is considered separately. We
G_{558} initialize n variables for accumulating the contributions to the quantities $\hat{T}_b(l)$
G_{559} according to (2). Then, for each forbidden set F separately, we compute the
G_{560} quantities $U_b(F, l, j, v)$ for all j and v, incrementally increasing $l = 1, 2, \ldots, n$.
G_{561} Since we need to remember only the entries for two consecutive values l at a
G_{562} time, this requires only $O(nk)$ space. Along the way, we add the contributions
G_{563} $U_b(F, l, k, b)$ to (2).

G_{564} In summary, we can calculate $\hat{T}_b(1), \hat{T}_b(2), \ldots$ in space $O(nk)$. Compared to
G_{565} the exponential-space algorithm, we need an additional factor k in the runtime,
G_{566} for the k choices of the parameter j.

B.4   Recovering the solution

G_{567}

G_{568} The algorithm, as described so far, works as an oracle that only gives a yes-no
G_{569} answer (and a length l), but it does not produce the solution. To recover the
G_{570} solution, we will call the oracle repeatedly with different inputs.

G_{571} Suppose the algorithm was successful in the sense that some number $\hat{T}_b(l)$
G_{572} turned out to be nonzero, after finding only zero values for all smaller values of l.
G_{573} By Lemma 1, we conclude that there exists a simple cycle through b satisfying
G_{574} the edge set constraints, possibly (with small probability) shorter than l.

G_{575} We will find this cycle by selectively deleting parts of the edges and recom-
G_{576} puting $\hat{T}_b(1), \hat{T}_b(2), \ldots, \hat{T}_b(l)$ for the reduced graph to see whether this graph
G_{577} still contains a solution. In this way, we will determine the successive edges of
G_{578} the cycle, and, as we shall see, we will know the cycle after at most $4n \log_2 \frac{4m}{n}$
G_{579} iterations.

G_{580} The first edge out of b is an edge of S_1, and thus we start by looking for
G_{581} the first edge among these edges. (The other edge incident to b, by which we
G_{582} eventually return to b, is not in S_1, and thus there is no confusion between the
G_{583} two edges incident to b.) In the general step, we have determined an initial part
G_{584} of the cycle up to some vertex u, and we locate the outgoing edge among the
G_{585} edges (u, v) incident to u, excluding the edge leading to u that we have already
G_{586} used.

G_{587} In general, for a vertex u of degree d_u, we have a set of $d_u - 1$ potential edges.
G_{588} We locate the correct edge by binary search: We split the potential edges into
G_{589} two equal parts, and query whether a cycle still exists when one or the other
G_{590} part is removed. It may turn out (with small probability) that none of the two
G_{591} subproblems yields a positive answer. In this case, we repeat the oracle with
G_{592} new random weights. Since the success probability is greater than $1/2$, we are
G_{593} guaranteed to have a positive answer after at most two trials, in expectation. (We
G_{594} mention that such repeated trials may be necessary only when the length l for
G_{595} which we are looking is not the shortest length of a feasible cycle. Otherwise one
G_{596} can show, using arguments from the proof in Section B.5, that the polynomial
G_{597} for the original problem is the sum of the polynomials for the two subproblems.
G_{598} Thus, at least one of the two subproblems must give a positive answer.) After at

G_{599} most $2\lceil \log_2(d_u - 1)\rceil$ successful queries, we have narrowed down the search to a
G_{600} single outgoing edge uv, and we continue at the next vertex v.
G_{601} For a walk W of length l, we use, in expectation, less than $Q := \sum_{u \in W} 4(1 +$
G_{602} $\log_2 d_u)$ queries, where $\sum_{u \in W} d_u \leq 2m$. Q can be bounded by

G_{603}
$$Q \leq 4l \left(1 + \log_2 \tfrac{2m}{l}\right) \leq 4n \left(1 + \log_2 \tfrac{2m}{n}\right) = 4n \log_2 \tfrac{4m}{n},$$

G_{604} as claimed.
G_{605} During this procedure, it may also turn out that a solution with fewer than
G_{606} l edges exists. In this case, we know that we must have been in the unlikely case
G_{607} that the original algorithm failed to produce a nonzero value for the shortest
G_{608} solution l. We simply adjust l to the smaller value and continue.
G_{609} Note that this procedure is guaranteed to produce a simple cycle, although
G_{610} not necessarily the shortest one. The algorithm selects a branch only when the
G_{611} corresponding polynomial is nonzero, which implies that a simple cycle exists in
G_{612} that branch.
G_{613} Some simplifications are possible. For edges that are known to belong to
G_{614} every solution (for example if some set S_i contains only one edge), we can assign
G_{615} unit weight, thus saving random bits, reducing the degree of the polynomial,
G_{616} and increasing the success probability. Edges that would close a loop can be
G_{617} discarded. When the graph is sparse and $m = O(n)$, we can simply try the
G_{618} $d_u - 1$ edges one at a time instead of performing binary search, at no cost in
G_{619} terms of the asymptotic runtime.

G_{620} ## B.5   Proof of Lemma 1

G_{621} (a) By assumption, there is a simple cycle among the walks whose weights are
G_{622} collected in $\hat{T}_b(l)$, and we easily see that the monomial corresponding to such a
G_{623} walk occurs with coefficient 1. Hence $\hat{T}_b(l)$ is not identically zero. By definition,
G_{624} $\hat{T}_b(l)$ is a sum of weights of walks of length l, and hence it is clear that it is
G_{625} homogeneous of degree l.
G_{626} (b) For the second statement of the lemma, we have to show that $\hat{T}_b(l)$ is zero
G_{627} if there is no simple walk of length l or shorter. Since the field has characteristic 2,
G_{628} it suffices to establish a matching among those closed walks that satisfy the edge
G_{629} set constraints but don't represent simple cycles. Let W be such a walk. We will
G_{630} map W to another walk $\phi(W)$ that uses the same multiset of edges, by reversing
G_{631} (flipping) the order of the edges of a subpath between two visits to the same
G_{632} vertex v. Our procedure closely follows Björklund et al. [4], but we correct an
G_{633} error in their description.
G_{634} An example of the procedure is shown in Figure 13. We look for the first
G_{635} vertex v that occurs several times on the walk. (During this whole procedure,
G_{636} the occurrence of b at the start of the walk is never considered.) We look at the
G_{637} piece $[v \ldots v]$ between the first and last occurrence of v and flip it. If the sequence
G_{638} of vertices in this piece is not a palindrome, we are done. Otherwise, we cut out
G_{639} this piece from the walk. The resulting walk will still visit an edge from each
G_{640} S_i because such an edge cannot be part of a palindrome, since it is visited only

$$W = W_0 = 12341565143234546178657 1 = 1[23415651432]345461786571$$
$$W_0' = W_1 = 1\underline{2}345461786571 = 123[454]61786571$$
$$W_1' = W_2 = 123\underline{4}61786571 = 1234[61786]571$$

$$W = 1234156514323454[61786]571$$
$$\phi(W) = 1234156514323454[68716]571$$

Fig. 13. Mapping a nonsimple cycle W to another cycle $\phi(W)$. The start vertex is $b = 1$.

$G6_{41}$ once. Since the resulting walk W' is shorter than l, and thus shorter than L, by
$G6_{42}$ assumption, it cannot be a simple cycle, and it must contain repeated vertices.

$G6_{43}$ We proceed with W' instead of W. Eventually we must find a piece $[v \ldots v]$
$G6_{44}$ that is not a palindrome. We flip it in the original walk W, and the result is
$G6_{45}$ the walk $\phi(W)$ to which W is matched. (The flipped piece $[v \ldots v]$ does not
$G6_{46}$ necessarily start at the first occurrence of v in the original sequence W, because
$G6_{47}$ such an occurrence might have been eliminated as part of a palindrome.[6])

$G6_{48}$ It is important to note that after cutting out a palindrome $[v \ldots v]$, all re-
$G6_{49}$ peated vertices must come *after* the vertex v. Hence, when the procedure is
$G6_{50}$ applied to $\phi(W)$, it will perform exactly the same sequence of operations until
$G6_{51}$ the last step, where it will flip $\phi(W)$ back to W.

$G6_{52}$ Since the first edge of the walk is unchanged, the condition that this edge
$G6_{53}$ must belong to S_1 is left intact.

$G6_{54}$ This concludes the proof of Lemma 1.   □

$G6_{55}$ ## B.6   Other approaches

$G6_{56}$ The original algorithm of Björklund et al. [4] considers simple paths through
$G6_{57}$ k specified *vertices* or (single) edges. The treatment of vertex visits leads to
$G6_{58}$ some complications: To make the argument for Lemma 1 valid, Björklund et
$G6_{59}$ al.'s definition of allowed walks had to explicitly forbid "palindromic visits" to
$G6_{60}$ the specified vertices, visiting the same edge twice in succession (condition P4).
$G6_{61}$ Consequently, the algorithm for accumulating weight of walks has to take care
$G6_{62}$ of this technicality.

$G6_{63}$ Another Monte Carlo FPT algorithm for the problem of finding a simple
$G6_{64}$ cycle through k specified vertices was given by Wahlström [17]. It runs in time
$G6_{65}$ $O(2^k \text{poly}(n))$ in a graph with n vertices. This algorithm uses interesting algebraic
$G6_{66}$ techniques, and it is also based on inclusion-exclusion, but it does not seem to
$G6_{67}$ extend to the case where one out of a *set* of vertices (or edges) has to be visited.

$G6_{68}$ [6] Here our procedure differs from the method described in [4], where the flipped sub-
$G6_{69}$ sequence extends between the first and last occurrence of v in W. In this form, the
$G6_{70}$ mapping ϕ is not an involution. The proof in [4], however, applies to the method as
       described here. This inconsistency was confirmed by the authors (Nina Taslaman,
$G6_{71}$ private communication, February 2021).

G672

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

C.1   Getting a primitive polynomial in quadratic time

G673 There is a completely naive algorithm for constructing $\mathbb{F}_q$ in $O(q^2)$ time and
G674 $O(q)$ space, for $q = 2^s$, assuming q fits in a word (s bits). Simply try out all
G675 polynomials $p(x)$ over $\mathbb{F}_2$ of degree less than s. There are q possibilities.
G676   For each $p(x)$, try to construct the logarithm table ("index table") in $O(q)$
G677 steps by trying whether the polynomial x generates the nonzero elements of
G678 $\mathbb{F}_2[x]$: Start with the string $00\ldots01$ representing the polynomial 1, and multiply
G679 by x by shifting to the left, and adding $p(x)$ (XOR with the corresponding bit
G680 string) to clear the highest bit if necessary. Repeat $q-2$ times and check whether
G681 $00\ldots01$ reappears. If not, then we are done.
G682   (This will be successful iff $p(x)$ is a primitive polynomial modulo 2. Actually,
G683 it is sufficient to check the powers x^k of x where k is a maximal proper divisor
G684 of $q - 1$. The primes dividing $q - 1$ can be trivially found in $O(\sqrt{q})$ time, and
G685 there are less than $\log q$ of them. Computing x^k takes $O(s^2 \log q) = O(\log^3 q)$
G686 time if the $O(s^2)$ schoolbook method of multiplication modulo $p(x)$ is applied,
G687 together with repeated squaring to get the power. Thus, $O(q \log^3 q)$ time instead
G688 of $O(q^2)$. Probably a gross overestimate because primitive polynomials modulo 2
G689 are frequent; there are $\phi(q - 1)/s$ of them.)

G690

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

G691 This is achieved by a degree-2 field extension. We look for an irreducible poly-
G692 nomial of the form $p(x) = x^2 + x + p_0$ over $\mathbb{F}_q$. If this polynomial were reducible,
G693 we could write

G694
$$p(x) = (x + a)(x + b) = x^2 + (a + b)x + ab$$

G695 with $a + b = 1$, hence $b = 1 + a$, and

G696
$$p(x) = (x + a)(x + (a + 1)) = x^2 + x + a(a + 1)$$

G697 The expression $a(a+1)$ can take at most $q/2$ different values, because $a = c$ and
G698 $a = c + 1$ lead to the same product, since $(c + 1) + 1 = c$. Thus, we can group
G699 the q potential values a into $q/2$ pairs with the same product, and at least $q/2$
G700 must be unused. We can find the range of $a(a + 1)$ by marking its values in an
G701 array of size q. Any unmarked value can be used as the constant term p_0. This
G702 takes $O(q)$ time and space.
G703   Multiplication of two polynomials $a_1x + a_0$ and $b_1x + b_0$ gives the product

G704
$$c_1 x + c_0 = (a_1 x + a_0)(b_1 x + b_0)$$

G705
$$= a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0$$

G706
$$= a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0 - a_1 b_1 (x^2 + x + p_0)$$

G707
$$= (a_1 b_0 + a_0 b_1 + a_1 b_1)x + (a_0 b_0 + a_1 b_1 p_0)$$

G_{708} Multiplication can therefore be carried out as follows:

$$c_1 = a_1 b_0 + a_0 b_1 + a_1 b_1$$
G_{709}

$$= (a_1 + a_0)(b_1 + b_0) - a_0 b_0$$
G_{710}

$$c_0 = a_0 b_0 + a_1 b_1 p_0$$
G_{711}

G_{712} with four multiplications in $\mathbb{F}_q$ (and four additions), using the common term
G_{713} $a_0 b_0$ for both coefficients.[7]

C.3   Setup of a finite field $\mathbb{F}_q$ with $q > n^2$ in deterministic $O(n)$
G_{714} time and space

G_{715} Let $C = \lceil (\log_2 n)/2 \rceil$. First we construct $\mathbb{F}_q$ for $q = 2^C = O(\sqrt{n})$ in $O(q^2) =$
G_{716} $O(n)$ time, as described in Section C.1. To carry out multiplications in $\mathbb{F}_q$ in
G_{717} constant time, we can store a logarithm and antilogarithm table, in $O(q)$ space,
G_{718} or we can even compute a complete multiplication table, in $O(q^2) = O(n)$ time
G_{719} and space.
G_{720} Then, by the method of Section C.2, we go from $q = 2^C$ to $q = 2^{2C}$, and
G_{721} finally from $q = 2^{2C}$ to $q = 2^{4C}$, in $O(2^{2C}) = O(n)$ time and space.
G_{722} Then addition and multiplication in $\mathbb{F}_{2^{4C}}$ can be carried out in constant
G_{723} time. Multiplication goes down two recursive levels, from $q = 2^{4C}$ via $q = 2^{2C}$
G_{724} to $q = 2^C$, before the 16 resulting multiplications are resolved by table look-up.
G_{725} If we prefer, we can eliminate the lower level of recursion by building a
G_{726} log/antilog table for $q = 2^{2C} = O(n)$ of size $O(n)$, to do the multiplication by
G_{727} table look-up already at this level. To prepare the tables, we need a generating
G_{728} element of $\mathbb{F}_{2^{2C}}$. Such a generating element can be constructed in $O(2^{2C}) = O(n)$
G_{729} time, as shown in the next section C.4.

C.4   Constructing an index table without knowing a generator
G_{730}

G_{731} We assume that multiplication in $\mathbb{F}_q$ takes constant time, but no generating
G_{732} element for the multiplicative group of $\mathbb{F}_q$ is given. Our goal is to construct
G_{733} logarithm and antilogarithm tables (index tables), of size $q - 1$, in time $O(q)$. In
G_{734} order to generate these tables, we need a generating element.
G_{735} Essentially, we have to find a generator of a cyclic group, which is given by
G_{736} a multiplication oracle and the list of its elements.
G_{737} We start with an arbitrary element $a_1 \neq 0$ and try to construct the index
G_{738} table by running through the powers of a_1. We mark the elements that we find.
G_{739} If the process returns to 1 before marking all nonzero elements, we pick an
G_{740} unmarked element a_2 and run the same process with a_2.

G_{741} [7] This is the same trick as for multiplying two complex numbers with three multiplica-
G_{742} tions instead of four. By contrast, [12, Section 6.8] propose irreducible polynomials
G_{743} of the form $x^2 + p_1 x + 1$, leading to five multiplications. On the other hand, in cases
G_{744} where multiplications are done by index tables, the mere number of multiplication
G_{745} operations does not determine the runtime alone; a common factor that appears in
      several multiplications saves lookup time in the logarithm tables.

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 $o = \mathrm{lcm}(o_1, o_2)$, 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 $\mathrm{lcm}(o_1, o_2)$. 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 $f_1 \leq f_2$ 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, $o = \mathrm{lcm}(o_1, o_2) = o_1 o_2$. Let g be a generator of the group $\langle a_1, a_2 \rangle = \{1, g, g^2, \ldots, g^{o-1}\}$. 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 Eulerian algorithm for calculating the greatest common divisor of o_1 and o_2, we find u_1, u_2 such that

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

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.

D   Adapting the Edge Set Constraints

In Section 4 we have outlined a modified construction of the graph G on which a nonintersecting Eulerian cycle is sought, which avoids the quadratic blow-up of the number of edges.

Here we give more details of this construction, see Figure 5c. We cut off each run of consecutive edges, like *fghi*, by an additional edge (shown dotted in Figure 5c) and place a single terminal node there.

One has to take care that the cycle that is found does not use two such terminal edges in succession, like the edges crossing f and h, because such a

$G785$ cycle would not correspond to a valid curve. This constraint must be added to
$G786$ the problem definition, and the recursion (1) must be modified accordingly.
$G787$ The sets S_i have a special structure. Each set S_i consists of a few vertex-
$G788$ disjoint edges (the thick edges in Figure 14), which we call *central edges*. We
$G789$ call the edges connecting the central edges to the remainder of the graph the
$G790$ *peripheral edges*. We don't want to consider cycles that use two such peripheral
$G791$ edges (of the same S_i) in succession, without going through the central edge. We
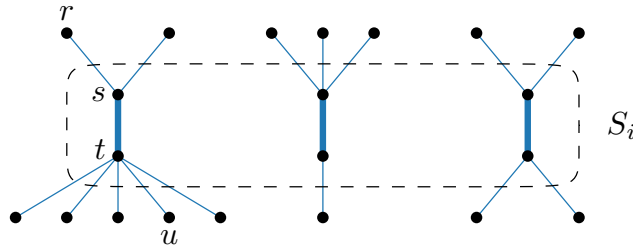$G792$ impose this as an extra condition on the walks whose weights we accumulate.

Fig. 14. A set S_i and its connecting edges

$G793$ It is easy to incorporate this condition in the recursion of Section 4.1: Instead
$G794$ of the quantities $T_b(R, l, v)$, we work with quantities $\tilde{T}_b(R, l, v, p)$ that depend
$G795$ on an additional parameter p. This is one bit that tells whether the last edge
$G796$ of the walk has traversed a peripheral edge in the direction towards the central
$G797$ edge. If this is the case, we force the walk to use the central edge in the next
$G798$ step.
$G799$ In this way, the additional condition incurs a blow-up of at most a factor 2
$G800$ in the size of the dynamic programming tables and in the runtime.
$G801$ We now argue that Lemma 1 still holds for these modified quantities. The
$G802$ proof in Section B.5 goes through for the following reason. The conditions on
$G803$ allowed walks ensure that an endpoint of a central edge, like the vertex s in
$G804$ Figure 14), is always part of a subpath consisting of a central edge surrounded
$G805$ by two peripheral edges, like (r, s, t, u) or (u, t, s, r). Such a subpath cannot be
$G806$ part of a palindrome, since (s, t) belongs to a special set S_i, and for the same
$G807$ reason, s can never be a repeated vertex of a walk. If the bijection constructed in
$G808$ the proof reverses a subpath that goes through the vertex s, this is no problem
$G809$ because the reversed traversal does not violate the extra condition.